Integer Arithmetic IP Cores User Guide

UG-01063
2014.12.19

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

Integer Arithmetic Megafunctions 1

2014.12.19

UG-01063 Subscribe Send Feedback

You can use Altera® integer megafunction IP cores to perform mathematical operations in your design. These functions offer more efficient logic synthesis and device implementation than coding your own functions. You can customize the IP cores to accommodate your design requirements.

Altera integer arithmetic megafunctions are divided into the following two categories:

- Library of parameterized modules (LPM) IP cores
- Altera-specific (ALT) IP cores

The following table lists the integer arithmetic IP cores.

Table 1-1: List of IP Cores

IP Cores	Function Overview
LPM Megafunctions	
LPM_COUNTER (Counter)	Counter
LPM_DIVIDE (Divider)	Divider
LPM_MULT (Multiplier)	Multiplier
Altera-specific (ALT) Megafunctions	
ALTECC	ECC Encoder/Decoder
ALTERA_MULT_ADD (Multiply-Adder)	Multiplier-Adder
ALTMEMMULT (Memory-based Constant Coefficient Multiplier)	Memory-based Constant Coefficient Multiplier
ALTMULT_ACCUM (Multiply-Accumulate)	Multiplier-Accumulator
ALTERA_MULT_ADD (Multiply-Adder)	Multiplier-Adder
ALTMULT_COMPLEX (Complex Multiplier)	Complex Multiplier
ALTSQRT (Integer Square Root)	Integer Square-Root
PARALLEL_ADD (Parallel Adder)	Parallel Adder

If you are unfamiliar with IP cores, refer to the **Introduction to IP Cores User Guide**.

Altera also provides floating-point IP cores. For more information about the floating-point IP cores, refer to the **Floating-Point IP Cores User Guide**.

Design Example Files

Altera provides design example files that are simulated in the ModelSim®-Altera software to generate a waveform display of the device behavior.

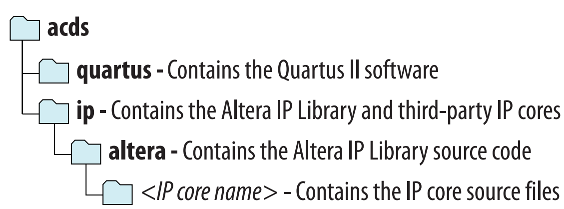
You should be familiar with the ModelSim-Altera software before using the design examples. To get started with the ModelSim-Altera software, refer to the **ModelSim-Altera Software Support** page on the Altera website. The support page includes links to such topics as installation, usage, and troubleshooting. For more details about the design example for a specific IP core, refer to the “Design Example” section for that megafunction.

Design examples are provided only for some IP cores in this user guide.

Installing and Licensing IP Cores

The Altera IP Library provides many useful IP core functions for production use without purchasing an additional license. You can evaluate any Altera® IP core in simulation and compilation in the Quartus® II software using the OpenCore® evaluation feature. Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. You can use the OpenCore Plus feature to evaluate IP that requires purchase of an additional license until you are satisfied with the functionality and performance. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product.

Figure 1-1: IP Core Installation Path



Note: The default IP installation directory on Windows is **<*drive*>:\altera**<*version number*>; on Linux it is <*home directory*>**/altera/** <*version number*>.

Related Information

- **Altera Licensing Site**
- **Altera Software Installation and Licensing Manual**

Customizing and Generating IP Cores

You can customize IP cores to support a wide variety of applications. The Quartus II IP Catalog and parameter editor allow you to quickly select and configure IP core ports, features, and output files.

IP Catalog and Parameter Editor

The Quartus II IP Catalog (**Tools** > **IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

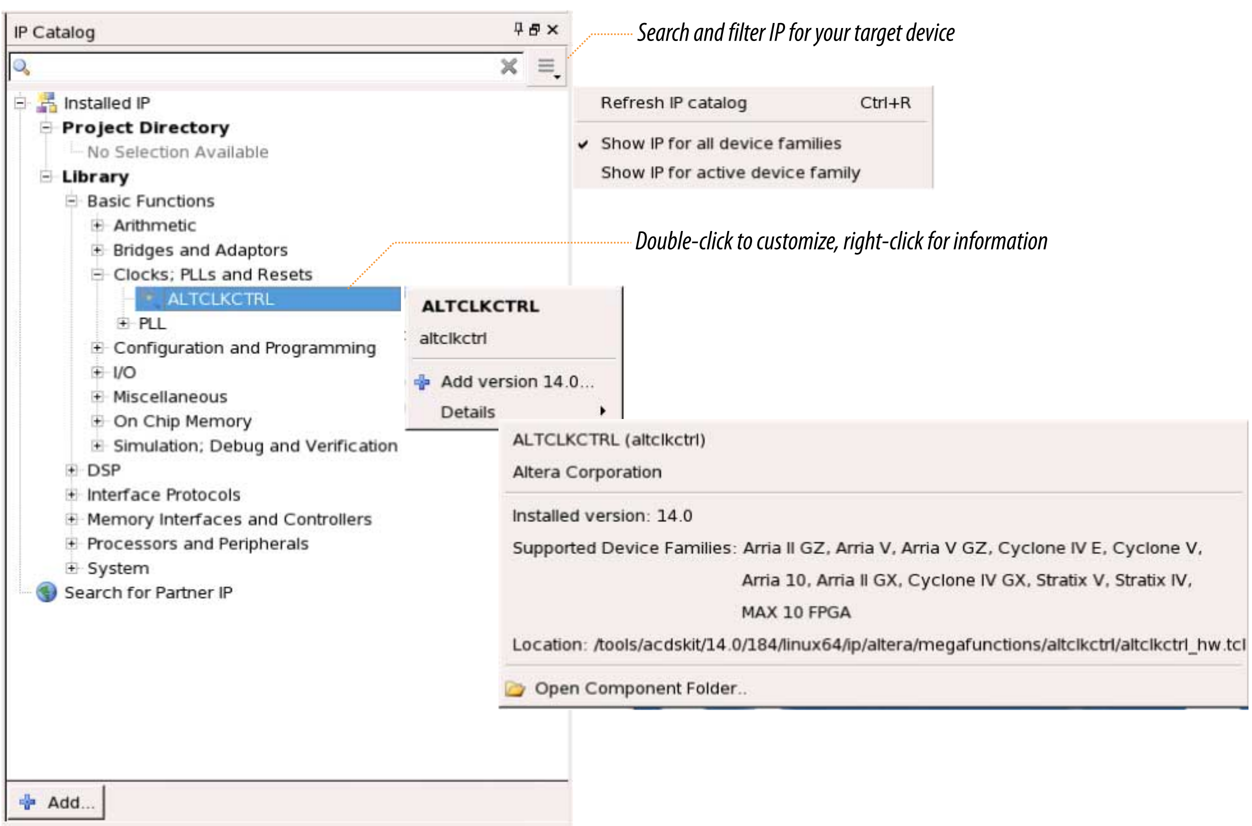
Note: The IP Catalog (**Tools** > **IP Catalog)** and parameter editor replace the MegaWizard™ Plug-In Manager for IP selection and parameterization, beginning in Quartus II software version 14.0. Use the IP Catalog and parameter editor to locate and paramaterize Altera IP cores.

The IP Catalog lists IP cores available for your design. Double-click any IP core to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Qsys system file (**.qsys**) or Quartus II IP file (**.qip**) representing the IP core in your project. You can also parameterize an IP variation without an open project.

Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**.
- Search to locate any full or partial IP core name in IP Catalog. Click **Search for Partner IP**, to access partner IP information on the Altera website.
- Right-click an IP core name in IP Catalog to display details about supported devices, open the IP core's installation folder, andor view links to documentation.

Figure 1-2: Quartus II IP Catalog



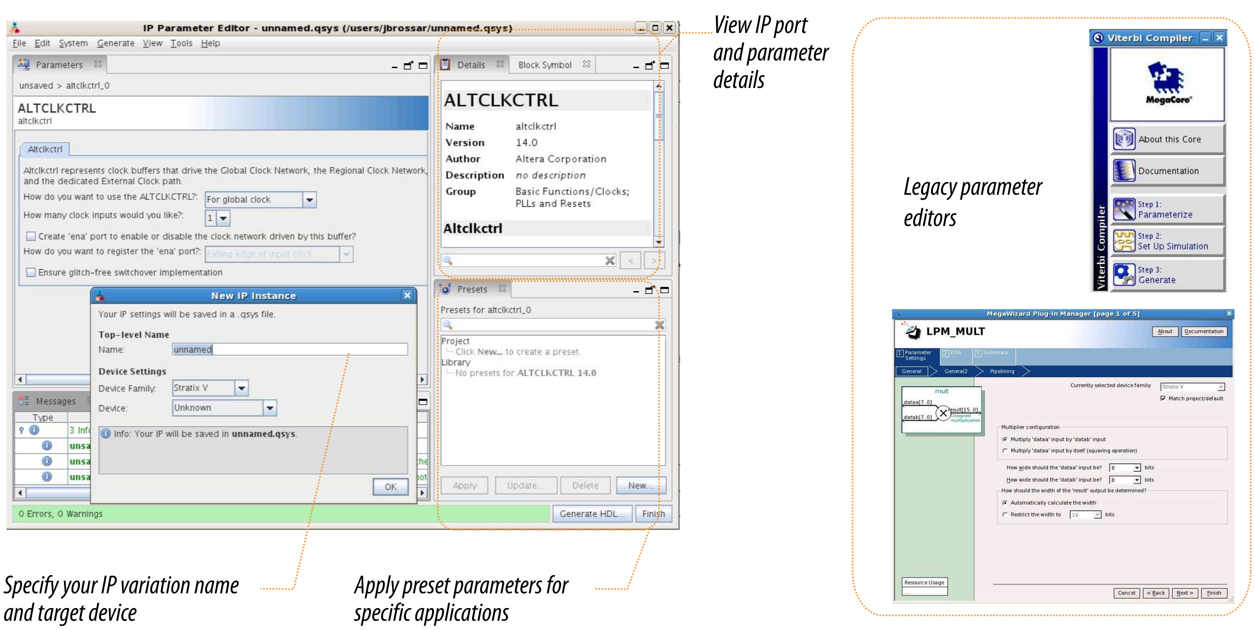
Note: The IP Catalog is also available in Qsys (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus II IP Catalog. For more information about using the Qsys IP Catalog, refer to *Creating a System with Qsys* in the *Quartus II Handbook*.

Using the Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options.

- Use preset settings in the parameter editor (where provided) to instantly apply preset parameter values for specific applications.
- View port and parameter descriptions, and links to documentation.
- Generate testbench systems or example designs (where provided).

Figure 1-3: IP Parameter Editors

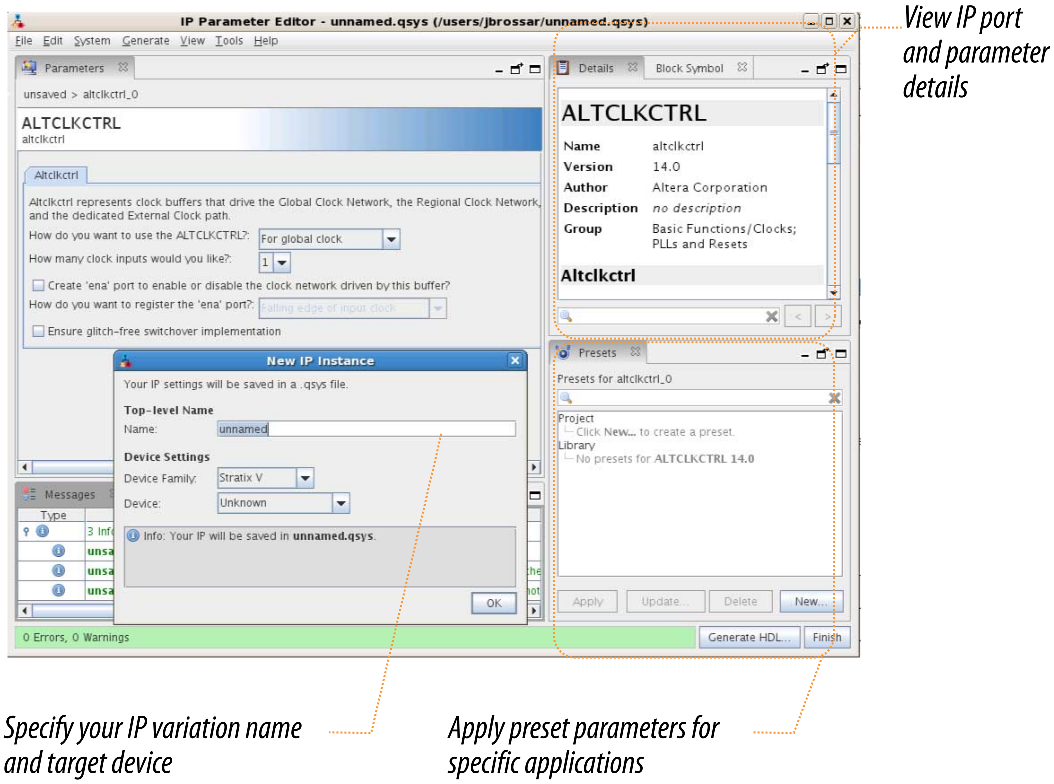


Specifying IP Core Parameters and Options

The parameter editor GUI allows you to quickly configure your custom IP variation. Use the following steps to specify IP core options and parameters in the Quartus II software. Refer to *Specifying IP Core Parameters and Options (Legacy Parameter Editors)* for configuration of IP cores using the legacy parameter editor.

1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. The parameter editor saves the IP variation settings in a file named <*your_ip*>**.qsys**. Click **OK**.
3. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following. Refer to your IP core user guide for information about specific IP core parameters.
    - Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
    - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
    - Specify options for processing the IP core files in other EDA tools.
4. Click **Generate HDL**, the **Generation** dialog box appears.
5. Specify output file generation options, and then click **Generate**. The IP variation files generate according to your specifications.
6. To generate a simulation testbench, click **Generate** > **Generate Testbench System**.
7. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate** > **HDL Example**.
8. Click **Finish**. The parameter editor adds the top-level **.qsys** file to the current project automatically. If you are prompted to manually add the **.qsys** file to the project, click **Project** > **Add/Remove Files in Project** to add the file.
9. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Figure 1-4: IP Parameter Editor

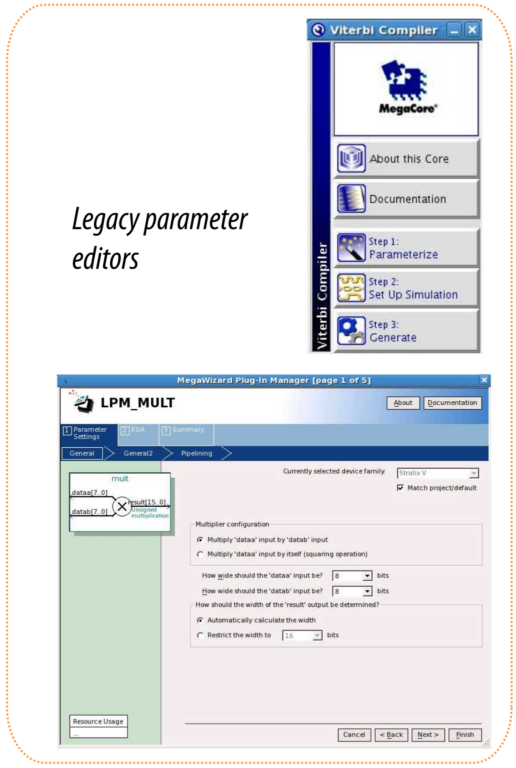


Specifying IP Core Parameters and Options (Legacy Parameter Editors)

Some IP cores use a legacy version of the parameter editor for configuration and generation. Use the following steps to configure and generate an IP variation using a legacy parameter editor.

Note: The legacy parameter editor generates a different output file structure than the latest parameter editor. Refer to *Specifying IP Core Parameters and Options* for configuration of IP cores that use the latest parameter editor.

Figure 1-5: Legacy Parameter Editors



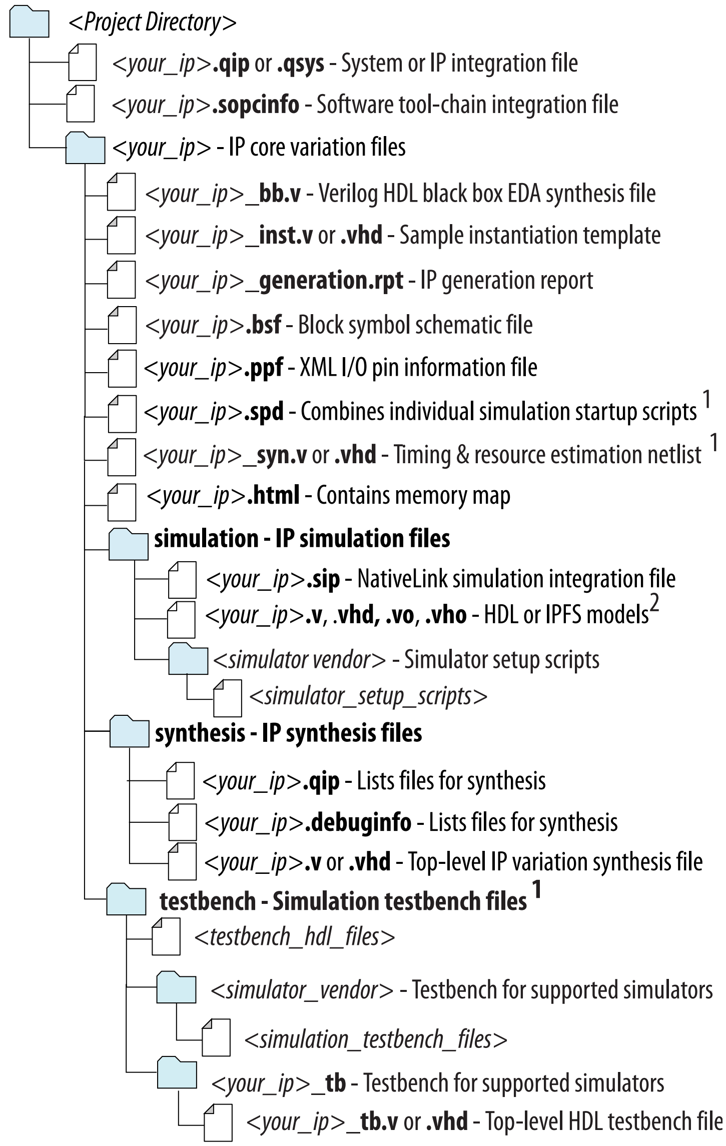
1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name and output HDL file type for your IP variation. This name identifies the IP core variation files in your project. Click **OK**.
3. Specify the parameters and options for your IP variation in the parameter editor. Refer to your IP core user guide for information about specific IP core parameters.
4. Click **Finish** or **Generate** (depending on the parameter editor version). The parameter editor generates the files for your IP variation according to your specifications. Click **Exit** if prompted when generation is complete. The parameter editor adds the top-level **.qip** file to the current project automatically.

   Note: To manually add an IP variation generated with legacy parameter editor to a project, click **Project** > **Add/Remove Files in Project** and add the IP variation **.qip** file.

Files Generated for Altera IP Cores (Legacy Parameter Editor)

The Quartus II software version generates the following output for your IP core that uses the legacy parameter editor.

Figure 1-6: IP Core Generated Files



Notes:
1. If supported and enabled for your IP variation
2. If functional simulation models are generated

Upgrading IP Cores

IP core variants generated with a previous version of the Quartus II software may require upgrading before use in the current version of the Quartus II software. Click **Project** > **Upgrade IP Components** to identify and upgrade IP core variants.

The **Upgrade IP Components** dialog box provides instructions when IP upgrade is required, optional, or unsupported for specific IP cores in your design. You must upgrade IP cores that require it before you can

compile the IP variation in the current version of the Quartus II software. Many Altera IP cores support automatic upgrade.

The upgrade process renames and preserves the existing variation file (**.v**, **.sv**, or **.vhd**) as <*my_variant*>_**BAK.v**, **.sv**, **.vhd** in the project directory.

Table 1-2: IP Core Upgrade Status

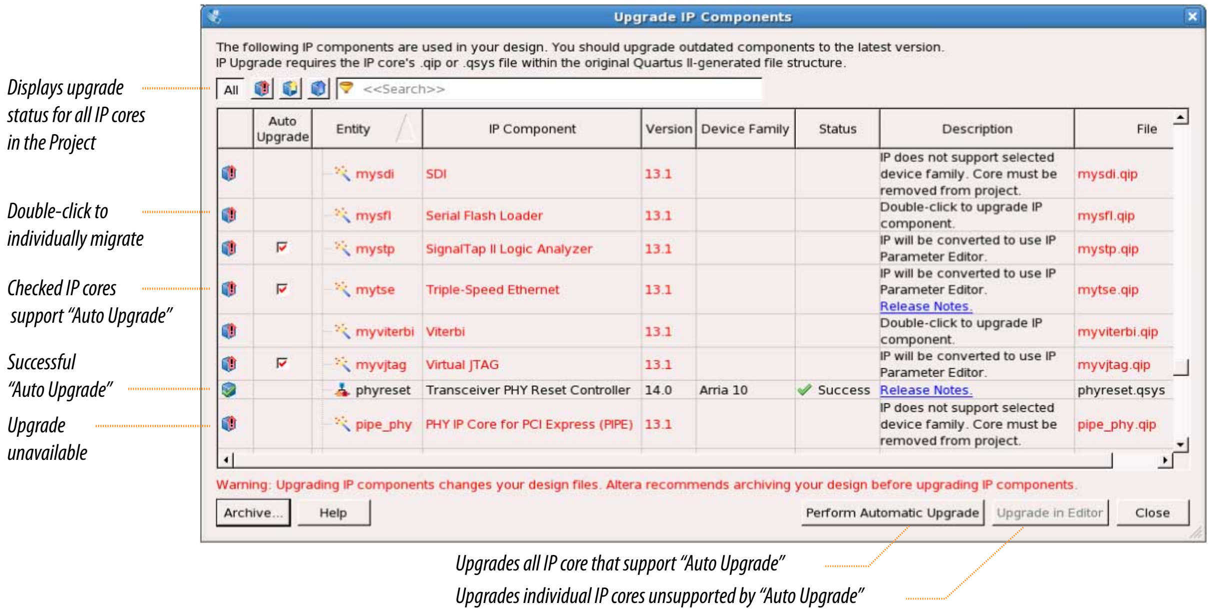
IP Core Status	Corrective Action
Required Upgrade IP Components	You must upgrade the IP variation before compiling in the current version of the Quartus II software.
Optional Upgrade IP Components	Upgrade is optional for this IP variation in the current version of the Quartus II software. You can upgrade this IP variation to take advantage of the latest development of this IP core. Alternatively you can retain previous IP core characteristics by declining to upgrade.
Upgrade Unsupported	Upgrade of the IP variation is not supported in the current version of the Quartus II software due to IP core end of life or incompatibility with the current version of the Quartus II software. You are prompted to replace the obsolete IP core with a current equivalent IP core from the IP Catalog.

Before you begin

- Archive the Quartus II project containing outdated IP cores in the original version of the Quartus II software: Click **Project** > **Archive Project** to save the project in your previous version of the Quartus II software. This archive preserves your original design source and project files.
- Restore the archived project in the latest version of the Quartus II software: Click **Project** > **Restore Archived Project**. Click **OK** if prompted to change to a supported device or overwrite the project database. File paths in the archive must be relative to the project directory. File paths in the archive must reference the IP variation **.v** or **.vhd** file or **.qsys** file (not the **.qip** file).

1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation. The **Upgrade IP Components** dialog automatically displays the status of IP cores in your project, along with instructions for upgrading each core. Click **Project** > **Upgrade IP Components** to access this dialog box manually.
2. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs provided with any Altera IP core regenerate automatically whenever you upgrade the IP core.

Figure 1-7: Upgrading IP Cores



Example 1-1: Upgrading IP Cores at the Command Line

You can upgrade IP cores that support auto upgrade at the command line. IP cores that do not support automatic upgrade do not support command line upgrade.

- To upgrade a single IP core that supports auto-upgrade, type the following command:

```
quartus_sh –ip_upgrade –variation_files <my_ip_filepath/my_ip>.<hdl>
<qii_project>

Example:
quartus_sh -ip_upgrade -variation_files mega/pll25.v hps_testx
```

- To simultaneously upgrade multiple IP cores that support auto-upgrade, type the following command:

```
quartus_sh –ip_upgrade –variation_files "<my_ip_filepath/my_ip1>.<hdl>;
<my_ip_filepath/my_ip2>.<hdl>"  <qii_project>

Example:
quartus_sh -ip_upgrade -variation_files "mega/pll_tx2.v;mega/pll3.v"
hps_testx
```

Note: IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *Altera IP Release Notes* reports any verification exceptions for Altera IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

Related Information

Altera IP Release Notes

Migrating IP Cores to a Different Device

IP migration allows you to target the latest device families with IP originally generated for a different device. Some Altera IP cores require individual migration to upgrade. The **Upgrade IP Components** dialog box prompts you to double-click IP cores that require individual migration.

1. To display IP cores requiring migration, click **Project** > **Upgrade IP Components**. The **Description** field prompts you to double-click IP cores that require individual migration.
2. Double-click the IP core name, and then click **OK** after reading the information panel.
   The parameter editor appears showing the original IP core parameters.
3. For the **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Finish**, and then click **Finish** again to migrate the IP variation using best-effort mapping to new parameters and settings. Click **OK** if you are prompted that the IP core is unsupported for the current device. A new parameter editor opens displaying best-effort mapped parameters.
5. Click **Generate HDL**, and then confirm the **Synthesis** and **Simulation** file options. Verilog is the parameter editor default HDL for synthesis files. If your original IP core was generated for VHDL, select **VHDL** to retain the original output HDL format.
6. To regenerate the new IP variation for the new target device, click **Generate**. When generation is complete, click **Close**.
7. Click **Finish** to complete migration of the IP core. Click **OK** if you are prompted to overwrite IP core files. The **Device Family** column displays the migrated device support. The migration process replaces *<my_ip>***.qip** with the *<my_ip>***.qsys** top-level IP file in your project.

   Note: If migration does not replace *<my_ip>***.qip** with *<my_ip>***.qsys**, click **Project > Add/Remove Files in Project** to replace the file in your project.
8. Review the latest parameters in the parameter editor or generated HDL for correctness. IP migration may change ports, parameters, or functionality of the IP core. During migration, the IP core's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If your upgraded IP core is represented by a symbol in a supporting Block Design File schematic, replace the symbol with the newly generated *<my_ip>***.bsf** after migration.

   Note: The migration process may change the IP variation interface, parameters, and functionality. This may require you to change your design or to re-parameterize your variant after the **Upgrade IP Components** dialog box indicates that migration is complete. The **Description** field identifies IP cores that require design or parameter changes.

Related Information

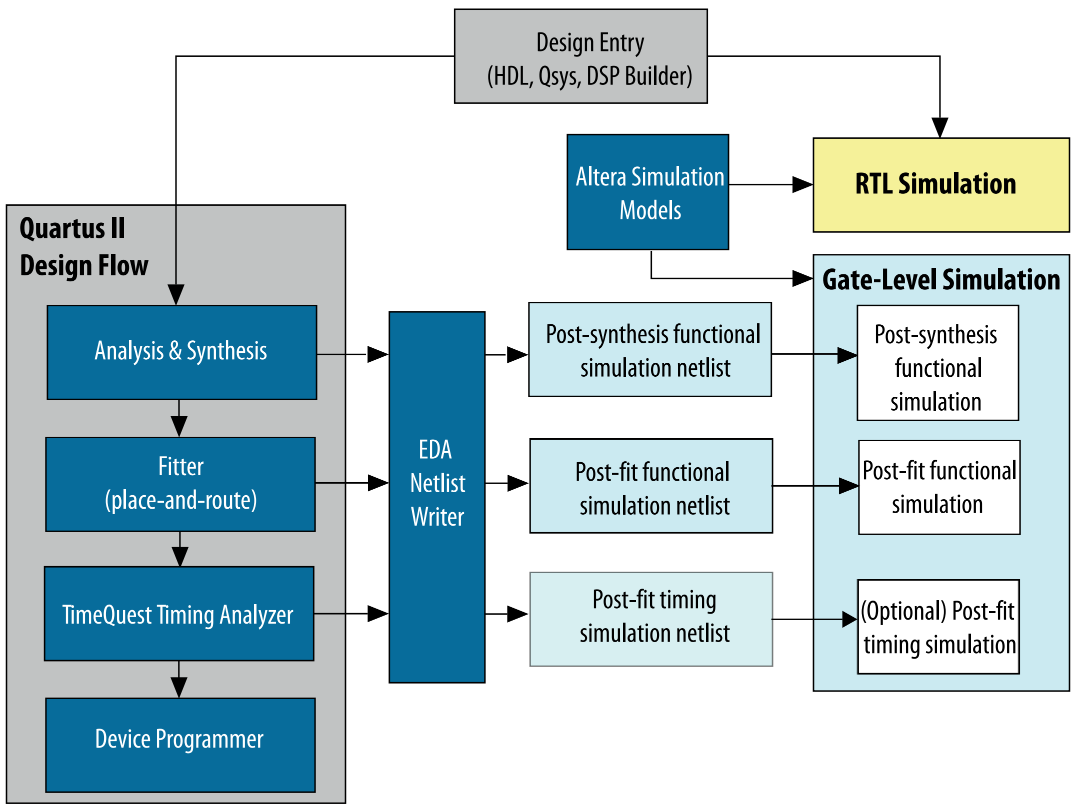
Altera IP Release Notes

Simulating Altera IP Cores in other EDA Tools

The Quartus II software supports RTL and gate-level design simulation of Altera IP cores in supported EDA simulators. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench. For a complete list of models or libraries required to simulate your IP core, refer to the scripts generated with the testbench. You can use the Quartus II NativeLink feature to automatically generate simulation files and scripts. NativeLink launches your preferred simulator from within the Quartus II software.

Figure 1-8: Simulation in Quartus II Design Flow



Note: Post-fit timing simulation is not supported for 28nm and later device archetectures. Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. These are all cycle-accurate models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some cores, only the plain text RTL model is generated, and you can simulate that model. Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

Related Information

Simulating Altera Designs

LPM_COUNTER (Counter) 2

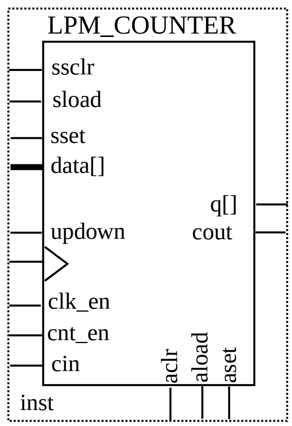
2014.12.19

UG-01063 Subscribe Send Feedback

The LPM_COUNTER megafunction is a binary counter that creates up counters, down counters and up or down counters with outputs of up to 256 bits wide.

The following figure shows the ports for the LPM_COUNTER megafunction.

Figure 2-1: LPM_COUNTER Ports



Features

The LPM_COUNTER megafunction offers the following features:

- Generates up, down, and up/down counters
- Generates the following counter types:
  - Plain binary— the counter increments starting from zero or decrements starting from 255
  - Modulus—the counter increments to or decrements from the modulus value specified by the user and repeats
- Supports optional synchronous clear, load, and set input ports
- Supports optional asynchronous clear, load, and set input ports
- Supports optional count enable and clock enable input ports
- Supports optional carry-in and carry-out ports

Resource Utilization and Performance

The following table provides resource utilization and performance information for the LPM_COUNTER megafunction.

Table 2-1: LPM_COUNTER Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			f_{MAX} (MHz)[1]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
Stratix III	4	-	9	4	6	723
	8	-	9	8	5	808
	16	-	17	16	9	705
	24	-	25	24	13	583
	32	-	33	32	17	489
	64	-	65	64	33	329
Stratix IV	4	-	9	4	6	768
	8	-	9	8	5	896
	16	-	17	16	9	825
	24	-	25	24	13	716
	32	-	33	32	17	639
	64	-	65	64	33	470

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **lpm.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module lpm_counter ( q, data, clock, cin, cout, clk_en, cnt_en, updown,
aset, aclr, aload, sset, sclr, sload, eq );
parameter lpm_type = "lpm_counter";
parameter lpm_width = 1;
parameter lpm_modulus = 0;
parameter lpm_direction = "UNUSED";
parameter lpm_avalue = "UNUSED";
parameter lpm_svalue = "UNUSED";
parameter lpm_pvalue = "UNUSED";
parameter lpm_port_updown = "PORT_CONNECTIVITY";
parameter lpm_hint = "UNUSED";
output [lpm_width-1:0] q;
```

(1) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

```
output cout;
output [15:0] eq;
input  cin;
input  [lpm_width-1:0] data;
input  clock, clk_en, cnt_en, updown;
input  aset, aclr, aload;
input  sset, sclr, sload;
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **LPM_PACK.vhd** in the *<Quartus II installation directory>***\libraries\vhdl\lpm** directory.

```
component LPM_MULT
        generic ( LPM_WIDTHA : natural;
                  LPM_WIDTHB : natural;
                  LPM_WIDTHS : natural := 1;
                  LPM_WIDTHP : natural;
LPM_REPRESENTATION : string := "UNSIGNED";
LPM_PIPELINE : natural := 0;
LPM_TYPE: string := L_MULT;
LPM_HINT : string := "UNUSED");
port ( DATAA : in std_logic_vector(LPM_WIDTHA-1 downto 0);
DATAB : in std_logic_vector(LPM_WIDTHB-1 downto 0);
ACLR : in std_logic := '0';
CLOCK : in std_logic := '0';
CLKEN : in std_logic := '1';
SUM : in std_logic_vector(LPM_WIDTHS-1 downto 0) := (OTHERS => '0');
RESULT : out std_logic_vector(LPM_WIDTHP-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY lpm;
USE lpm.lpm_components.all;
```

Ports

The following tables list the input and output ports for the LPM_COUNTER megafunction.

Table 2-2: LPM_COUNTER Megafunction Input Ports

Port Name	Required	Description
`data[]`	No	Parallel data input to the counter. The size of the input port depends on the `LPM_WIDTH` parameter value.
`clock`	Yes	Positive-edge-triggered clock input.
`clk_en`	No	Clock enable input to enable all synchronous activities. If omitted, the default value is `1`.

Port Name	Required	Description
cnt_en	No	Count enable input to disable the count when asserted low without affecting sload, sset, or sclr. If omitted, the default value is 1.
updown	No	Controls the direction of the count. When asserted high (1), the count direction is up, and when asserted low (0), the count direction is down. If the LPM_DIRECTION parameter is used, the updown port cannot be connected. If LPM_DIRECTION is not used, the updown port is optional. If omitted, the default value is up (1).
cin	No	Carry-in to the low-order bit. For up counters, the behavior of the cin input is identical to the behavior of the cnt_en input. If omitted, the default value is 1 (VCC).
aclr	No	Asynchronous clear input. If both aset and aclr are used and asserted, aclr overrides aset. If omitted, the default value is 0 (disabled).
aset	No	Asynchronous set input. Specifies the q[] outputs as all 1s, or to the value specified by the LPM_AVALUE parameter. If both the aset and aclr ports are used and asserted, the value of the aclr port overrides the value of the aset port. If omitted, the default value is 0, disabled.
aload	No	Asynchronous load input that asynchronously loads the counter with the value on the data input. When the aload port is used, the data[] port must be connected. If omitted, the default value is 0, disabled.
sclr	No	Synchronous clear input that clears the counter on the next active clock edge. If both the sset and sclr ports are used and asserted, the value of the sclr port overrides the value of the sset port. If omitted, the default value is 0, disabled.
sset	No	Synchronous set input that sets the counter on the next active clock edge. Specifies the value of the q outputs as all 1s, or to the value specified by the LPM_SVALUE parameter. If both the sset and sclr ports are used and asserted, the value of the sclr port overrides the value of the sset port. If omitted, the default value is 0 (disabled).
sload	No	Synchronous load input that loads the counter with data[] on the next active clock edge. When the sload port is used, the data[] port must be connected. If omitted, the default value is 0 (disabled).

Table 2-3: LPM_COUNTER Megafunction Output Ports

Port Name	Required	Description
q[]	No	Data output from the counter. The size of the output port depends on the LPM_WIDTH parameter value. Either q[] or at least one of the eq[15..0] ports must be connected.

Port Name	Required	Description
`eq[15..0]`	No	Counter decode output. The `eq[15..0]` port is not accessible using the MegaWizard Plug-In Manager as it is for AHDL use only. Either the `q[]` port or `eq[]` port must be connected. Up to `c eq` ports can be used (`0` <= `c` <= `15`). Only the 16 lowest count values are decoded. When the count value is `c`, the `eqc` output is asserted high (`1`). For example, when the count is `0`, `eq0` = `1`, when the count is `1`, `eq1` = `1`, and when the count is `15`, `eq 15` = `1`. Decoded output for count values of 16 or greater require external decoding. The `eq[15..0]` outputs are asynchronous to the `q[]` output.
`cout`	No	Carry-out port of the counter's MSB bit. It can be used to connect to another counter to create a larger counter.

Parameters

The following table lists the parameters for the LPM_COUNTER megafunction.

Table 2-4: LPM_COUNTER Megafunction Parameters

Parameter Name	Type	Required	Description
`LPM_WIDTH`	Integer	Yes	Specifies the widths of the `data[]` and `q[]` ports, if they are used.
`LPM_DIRECTION`	String	No	Values are `UP`, `DOWN`, and `UNUSED`. If the `LPM_DIRECTION` parameter is used, the `updown` port cannot be connected. When the `updown` port is not connected, the `LPM_DIRECTION` parameter default value is `UP`.
`LPM_MODULUS`	Integer	No	The maximum count, plus one. Number of unique states in the counter's cycle. If the load value is larger than the `LPM_MODULUS` parameter, the behavior of the counter is not specified.
`LPM_AVALUE`	Integer/ String	No	Constant value that is loaded when `aset` is asserted high. If the value specified is larger than or equal to `<modulus>`, the behavior of the counter is an undefined (X) logic level, where `<modulus>` is `LPM_MODULUS`, if present, or 2 ^ `LPM_WIDTH`. Altera recommends that you specify this value as a decimal number for AHDL designs.
`LPM_SVALUE`	Integer/ String	No	Constant value that is loaded on the rising edge of the clock port when the `sset` port is asserted high. Altera recommends that you specify this value as a decimal number for AHDL designs.

Parameter Name	Type	Required	Description
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES" The default value is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the LPM_COUNTER megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
CARRY_CNT_EN	String	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the CARRY_CNT_EN parameter in VHDL design files. Values are SMART, ON, OFF, and UNUSED. Enables the LPM_COUNTER function to propagate the cnt_en signal through the carry chain. In some cases, the CARRY_CNT_EN parameter setting might have a slight impact on the speed, so you might want to turn it off. The default value is SMART, which provides the best trade-off between size and speed.
LABWIDE_SCLR	String	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the LABWIDE_SCLR parameter in VHDL design files. Values are ON, OFF, or UNUSED. The default value is ON. Allows you to disable the use of the LAB-wide sclr feature found in obsoleted device families. Turning this option off increases the chances of fully using the partially filled LABs, and thus may allow higher logic density when SCLR does not apply to a complete LAB. This parameter is available for backward compatibility, and Altera recommends you not to use this parameter.

Parameter Name	Type	Required	Description
LPM_PORT_UPDOWN	String	No	Specifies the usage of the updown input port. If omitted the default value is PORT_CONNECTIVITY. When the port value is set to PORT_USED, the port is treated as used. When the port value is set to PORT_UNUSED, the port is treated as unused. When the port value is set to PORT_CONNECTIVITY, the port usage is determined by checking the port connectivity.

LPM_DIVIDE (Divider) 3

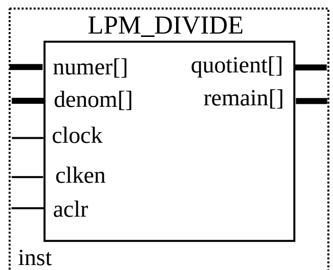
2014.12.19

UG-01063 Subscribe Send Feedback

The LPM_DIVIDE megafunction implements a divider to divide a numerator input value by a denominator input value to produce a quotient and a remainder.

The following figure shows the ports for the LPM_DIVIDE megafunction.

Figure 3-1: LPM_DIVIDE Ports



Features

The LPM_DIVIDE megafunction offers the following features:

- Generates a divider that divides a numerator input value by a denominator input value to produce a quotient and a remainder.
- Supports data width of 1–256 bits.
- Supports signed and unsigned data representation format for both the numerator and denominator values.
- Supports area or speed optimization.
- Provides an option to specify a positive remainder output.
- Supports pipelining configurable output latency.
- Supports optional asynchronous clear and clock enable ports.

Resource Utilization and Performance

The following table provides resource utilization and performance information for the LPM_DIVIDE megafunction.

Table 3-1: LPM_DIVIDE Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
Stratix III	10	1	131	0	70	133
	30	5	1017	0	635	71
	64	10	4345	0	2623	41
Stratix IV	10	1	131	0	70	138
	30	5	1018	0	642	82
	64	10	4347	0	2634	48

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **lpm.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module lpm_divide ( quotient, remain, numer, denom, clock, clken, aclr);
parameter lpm_type = "lpm_divide";
parameter lpm_widthn = 1;
parameter lpm_widthd = 1;
parameter lpm_nrepresentation = "UNSIGNED";
parameter lpm_drepresentation = "UNSIGNED";
parameter lpm_remainderpositive = "TRUE";
parameter lpm_pipeline = 0;
parameter lpm_hint = "UNUSED";
input  clock;
input  clken;
input  aclr;
input  [lpm_widthn-1:0] numer;
input  [lpm_widthd-1:0] denom;
output [lpm_widthn-1:0] quotient;
output [lpm_widthd-1:0] remain;
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **LPM_PACK.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\lpm** directory.

```
component LPM_DIVIDE
        generic (LPM_WIDTHN : natural;
                 LPM_WIDTHD : natural;
LPM_NREPRESENTATION : string := "UNSIGNED";
LPM_DREPRESENTATION : string := "UNSIGNED";
LPM_PIPELINE : natural := 0;
LPM_TYPE : string := L_DIVIDE;
LPM_HINT : string := "UNUSED");
port (NUMER : in std_logic_vector(LPM_WIDTHN-1 downto 0);
```

```
DENOM : in std_logic_vector(LPM_WIDTHD-1 downto 0);
ACLR : in std_logic := '0';
CLOCK : in std_logic := '0';
CLKEN : in std_logic := '1';
QUOTIENT : out std_logic_vector(LPM_WIDTHN-1 downto 0);
REMAIN : out std_logic_vector(LPM_WIDTHD-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY lpm;
USE lpm.lpm_components.all;
```

Ports

The following tables list the input and output ports for the LPM_DIVIDE megafunction.

Table 3-2: LPM_DIVIDE Megafunction Input Ports

Port Name	Required	Description
`numer[]`	Yes	Numerator data input. The size of the input port depends on the `LPM_WIDTHN` parameter value.
`denom[]`	Yes	Denominator data input. The size of the input port depends on the `LPM_WIDTHD` parameter value.
`clock`	No	Clock input for pipelined usage. For `LPM_PIPELINE` values other than `0` (default), the clock port must be enabled.
`clken`	No	Clock enable pipelined usage. When the `clken` port is asserted high, the division operation takes place. When the signal is low, no operation occurs. If omitted, the default value is `1`.
`aclr`	No	Asynchronous clear port used at any time to reset the pipeline to all '`0`'s asynchronously to the clock input.

Table 3-3: LPM_DIVIDE Megafunction Output Ports

Port Name	Required	Description
`quotient[]`	Yes	Data output. The size of the output port depends on the `LPM_WIDTHN` parameter value.
`remain[]`	Yes	Data output. The size of the output port depends on the `LPM_WIDTHD` parameter value.

Parameters

The following table lists the parameters for the LPM_DIVIDE megafunction.

Parameter Name	Type	Required	Description
`LPM_WIDTHN`	Integer	Yes	Specifies the widths of the `numer[]` and `quotient[]` ports. Values are `1` to `64`.
`LPM_WIDTHD`	Integer	Yes	Specifies the widths of the `denom[]` and `remain[]` ports. Values are `1` to `64`.
`LPM_NREPRESENTATION`	String	No	Sign representation of the numerator input. Values are `SIGNED` and `UNSIGNED`. When this parameter is set to `SIGNED`, the divider interprets the `numer[]` input as signed two's complement.
`LPM_DREPRESENTATION`	String	No	Sign representation of the denominator input. Values are `SIGNED` and `UNSIGNED`. When this parameter is set to `SIGNED`, the divider interprets the `denom[]` input as signed two's complement.
`LPM_TYPE`	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files (**.vhd**).
`LPM_HINT`	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the `LPM_HINT` parameter to specify an Altera-specific parameter. For example: `LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES"` The default value is `UNUSED`.

Parameter Name	Type	Required	Description
LPM_REMAINDERPOSITIVE	String	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the LPM_REMAINDERPOSITIVE parameter in VHDL design files. Values are TRUE or FALSE. If this parameter is set to TRUE, then the value of the remain[] port must be greater than or equal to zero. If this parameter is set to TRUE, then the value of the remain[] port is either zero, or the value is the same sign, either positive or negative, as the value of the numer port. In order to reduce area and improve speed, Altera recommends setting this parameter to TRUE in operations where the remainder must be positive or where the remainder is unimportant.
MAXIMIZE_SPEED	Integer	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the MAXIMIZE_SPEED parameter in VHDL design files. Values are [0..9]. If used, the Quartus II software attempts to optimize a specific instance of the LPM_DIVIDE function for speed rather than routability, and overrides the setting of the Optimization Technique logic option. If MAXIMIZE_SPEED is unused, the value of the Optimization Technique option is used instead. If the value of MAXIMIZE_SPEED is 6 or higher, the Compiler optimizes the LPM_DIVIDE megafunctions for higher speed by using carry chains; if the value is 5 or less, the compiler implements the design without carry chains.
LPM_PIPELINE	Integer	No	Specifies the number of clock cycles of latency associated with the quotient[] and remain[] outputs. A value of zero (0) indicates that no latency exists, and that a purely combinational function is instantiated. If omitted, the default value is 0 (non-pipelined). You cannot specify a value for the LPM_PIPELINE parameter that is higher than LPM_WIDTHN.

Parameter Name	Type	Required	Description
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the LPM_DIVIDE megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
SKIP_BITS	Integer	No	Allows for more efficient fractional bit division to optimize logic on the leading bits by providing the number of leading GND to the LPM_DIVIDE megafunction. Specify the number of leading GND on the quotient output to this parameter.

LPM_MULT (Multiplier) 4

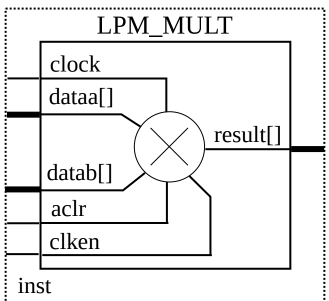
2014.12.19

UG-01063 Subscribe Send Feedback

The LPM_MULT megafunction implements a multiplier to multiply two input data values to produce a product as an output.

The following figure shows the ports for the LPM_MULT megafunction.

Figure 4-1: LPM_Mult Ports



Features

The LPM_MULT megafunction offers the following features:

- Generates a multiplier that multiplies two input data values
- Supports data width of 1–256 bits
- Supports signed and unsigned data representation format
- Supports area or speed optimization
- Supports pipelining with configurable output latency
- Provides an option for implementation in dedicated digital signal processing (DSP) block circuitry or logic elements (LEs)

  Note: When building multipliers larger than the natively supported size there may/will be a performance impact resulting from the cascading of the DSP blocks.
- Supports optional asynchronous clear and clock enable input ports

Resource Utilization and Performance

The LPM_MULT megafunction can be implemented using either logic resources or dedicated multiplier circuitry in Altera devices. Typically, the LPM_MULT megafunction is translated to the dedicated multiplier circuitry when it is available because it provides better performance and resource utilization. If all of the input data widths are smaller than or equal to nine bits, the function uses the 9 × 9 multiplier configuration in the dedicated multiplier. Otherwise, 18 × 18 multipliers are used to process data with widths between 10 bits and 18 bits.

For information about the architecture of the DSP blocks and embedded multipliers, and for detailed information about the hardware conversion process, refer to the DSP block and embedded multiplier chapters in the Stratix device series, Stratix II, Stratix III, and Cyclone II handbooks on the **Literature and Technical Documentation** page.

The following table provides resource utilization and performance information for the LPM_MULT megafunction.

Table 4-1: LPM_MULT Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			18-bit DSP	f_{MAX} (MHz)(2)
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)		
Stratix III	8 × 8	0	0	0	0	1	N/A
	16 × 16	0	0	0	0	2	
	32 × 32	0	0	0	0	4	
	16 × 16	3	0	0	0	2	645
	32 × 32	3	0	0	0	4	454
	64 × 64	3	92	128	82	16	191

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **lpm.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module lpm_mult ( result, dataa, datab, sum, clock, clken, aclr )
parameter lpm_type = "lpm_mult";
parameter lpm_widtha = 1;
parameter lpm_widthb = 1;
parameter lpm_widths = 1;
parameter lpm_widthp = 1;
parameter lpm_representation  = "UNSIGNED";
parameter lpm_pipeline  = 0;
parameter lpm_hint = "UNUSED";
input  clock;
input  clken;
input  aclr;
input  [lpm_widtha-1:0] dataa;
input  [lpm_widthb-1:0] datab;
```

(2) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

```
input  [lpm_widths-1:0] sum;
output [lpm_widthp-1:0] result;
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **LPM_PACK.vhd** in the *<Quartus II installation directory>***\libraries\vhdl\lpm** directory.

```
component LPM_MULT
        generic ( LPM_WIDTHA : natural;
                  LPM_WIDTHB : natural;
                  LPM_WIDTHS : natural := 1;
                  LPM_WIDTHP : natural;
LPM_REPRESENTATION : string := "UNSIGNED";
LPM_PIPELINE : natural := 0;
LPM_TYPE: string := L_MULT;
LPM_HINT : string := "UNUSED");
port ( DATAA : in std_logic_vector(LPM_WIDTHA-1 downto 0);
DATAB : in std_logic_vector(LPM_WIDTHB-1 downto 0);
ACLR : in std_logic := '0';
CLOCK : in std_logic := '0';
CLKEN : in std_logic := '1';
SUM : in std_logic_vector(LPM_WIDTHS-1 downto 0) := (OTHERS => '0');
RESULT : out std_logic_vector(LPM_WIDTHP-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY lpm;
USE lpm.lpm_components.all;
```

LPM_MULT Ports

Table 4-2: LPM_MULT IP Core Input Ports

Port Name	Required	Description
`dataa[]`	Yes	Data input. The size of the input port depends on the `LPM_WIDTHA` parameter value.
`datab[]`	Yes	Data input. The size of the input port depends on the `LPM_WIDTHB` parameter value.
`clock`	No	Clock input for pipelined usage. For `LPM_PIPELINE` values other than `0` (default), the clock port must be enabled.
`clken`	No	Clock enable for pipelined usage. When the `clken` port is asserted high, the adder/subtractor operation takes place. When the signal is low, no operation occurs. If omitted, the default value is `1`.

Port Name	Required	Description
aclr	No	Asynchronous clear port used at any time to reset the pipeline to all 0s, asynchronously to the clock signal. The pipeline initializes to an undefined (X) logic level. The outputs are a consistent, but non-zero value.

Table 4-3: LPM_MULT IP Core Output Ports

Port Name	Required	Description
result[]	Yes	Data output. The size of the output port depends on the LPM_WIDTHP parameter value. If LPM_WIDTHP < max (LPM_WIDTHA + LPM_WIDTHB, LPM_WIDTHS) or (LPM_WIDTHA + LPM_WIDTHS), only the LPM_WIDTHP MSBs are present.

LPM_MULT Parameters

The following table lists the parameters for the LPM_MULT megafunction.

Table 4-4: LPM_MULT Megafunction Parameters

Parameter Name	Type	Required	Description
LPM_WIDTHA	Integer	Yes	Specifies the width of the dataa[] port.
LPM_WIDTHB	Integer	Yes	Specifies the width of the datab[] port.
LPM_WIDTHP	Integer	Yes	Specifies the width of the result[] port.
LPM_REPRESENTATION	String	No	Specifies the type of multiplication performed. Values are SIGNED and UNSIGNED. If omitted, the default value is UNSIGNED. When this parameter value is set to SIGNED, the multiplier interprets the data input as signed two's complement.
LPM_PIPELINE	String	No	Specifies the number of latency clock cycles associated with the result[] output. A value of zero (0) indicates that no latency exists, and that a purely combinational function will be instantiated. For Stratix and Stratix GX devices, if the design uses DSP blocks, you can increase the performance of the design when the value of the LPM_PIPELINE parameter is 3 or less.

Parameter Name	Type	Required	Description
INPUT_A_IS_CONSTANT	String	No	You must use the LPM_HINT parameter to specify the INPUT_A_IS_CONSTANT parameter in VHDL design files. Values are YES, NO, and UNUSED. If dataa [] is connected to a constant value, setting INPUT_A_IS_CONSTANT to YES optimizes the multiplier for resource usage and speed. If omitted, the default value is NO.
INPUT_B_IS_CONSTANT	String	No	You must use the LPM_HINT parameter to specify the INPUT_B_IS_CONSTANT parameter in VHDL design files. Values are YES, NO, and UNUSED. If datab[] is connected to a constant value, setting INPUT_B_IS_CONSTANT to YES optimizes the multiplier for resource usage and speed. The default value is NO.
USE_EAB	String	No	Specifies RAM block usage. Values are ON and OFF. Setting the USE_EAB parameter to ON allows the Quartus II software to use embedded array blocks (EABs) to implement 4 x 4 or (8 x const value) building blocks in some obsolete devices. Altera recommends that you set USE_EAB to ON only when LCELLS are in short supply. This parameter is not available for simulation with other EDA simulators. If you wish to use this parameter when you instantiate the function in a Block Design File (**.bdf**), you must specify it by entering the parameter name and value manually with the **Parameters** tab in the **Symbol Properties** dialog box or in the **Block Properties** dialog box. You can also use this parameter name in a Text Design File (**.tdf**) or a Verilog Design File (**.v**). You must use the LPM_HINT parameter to specify the USE_EAB parameter in VHDL design files.

Parameter Name	Type	Required	Description
MAXIMIZE_SPEED	Integer	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the MAXIMIZE_SPEED parameter in VHDL design files. You can specify a value between 0 and 10. If used, the Quartus II software attempts to optimize a specific instance of the LPM_MULT function for speed rather than area, and overrides the setting of the **Optimization Technique** logic option. If MAXIMIZE_SPEED is unused, the value of the **Optimization Technique** option is used instead. For a SIGNED multiplier with no inputs being a constant, if the setting for MAXIMIZE_SPEED is 9-10, the Compiler optimizes the LPM_MULT megafunction for larger area. These settings are for backward compatibility only. If the setting is between 6-8, the Compiler optimizes for larger area and higher speed. If the setting is between 1-5, the Compiler optimizes for smaller area and high speed. If the setting is 0, the smallest and, generally, slowest design results. For designs with LPM_WIDTHB parameters that are non-power-of-2, the default setting is 1-5. For designs with LPM_WIDTHB parameters that are a power-of-2, the default value is 6-8. For an UNSIGNED multiplier with no inputs being a constant, if the setting for MAXIMIZE_SPEED is 6 or higher, the Compiler optimizes for larger area and higher speed. If the setting is 0 up to 5, which is the default value, the Compiler optimizes for smaller area. Note that specifying a value for MAXIMIZE_SPEED has an effect only if LPM_REPRESENTATION is set to SIGNED.
DEDICATED_MULTIPLIER_CIRCUITRY	String	No	Specifies whether to use the default dedicated multiplier circuitry implementation. Values are AUTO, YES, NO, and FIRM. If omitted, the default value is AUTO. For Stratix and Stratix GX devices, the value of AUTO specifies that the Quartus II software determines whether to use the dedicated multiplier circuitry based on the multiplier width. If a device does not have dedicated multiplier circuitry, the DEDICATED_MULTIPLIER_CIRCUITRY parameter has no effect and the value defaults to NO.

Parameter Name	Type	Required	Description
DSP_BLOCK_BALANCING	String	No	Specifies whether to use a dedicated multiplier circuitry implementation. Values are UNUSED, AUTO, DSP BLOCKS, and LOGIC ELEMENTS. If omitted, the default value is UNUSED. This parameter is available for all Altera devices except Cyclone, HardCopy, MAX II, MAX 3000, and MAX 7000 devices.
LOGIC_ELEMENTS	String	No	Specifies whether to use a logic element implementation based on the selected device family. When implemented in LEs, the LPM_MULT megafunction uses a variation on the Booth algorithm for all device families. Values are OFF, SIMPLE 18-BIT MULTIPLIERS, SIMPLE MULTIPLIERS, WIDTH 18-BIT MULTIPLIERS, and LOGIC ELEMENTS.
DEDICATED_MULTIPLIER_ MIN_INPUT_WIDTH_FOR_AUTO	Integer	No	Altera-specific parameter. You must use the LPM_HINT parameter to specify the DEDICATED_MULTIPLIER_MIN_ OUTPUT_WIDTH_FOR_AUTO parameter in VHDL design files. If the DEDICATED_ MULTIPLIER_CIRCUITRY parameter setting is AUTO, this parameter specifies the minimum value of the sum of the LPM_ WIDTHA and LPM_WIDTHB parameters in order for the multiplier to be built using dedicated circuitry.
INPUT_A_FIXED_VALUE	String	No	Specifies the value for the dataa[] port. This parameter is used when the INPUT_A_ IS_CONSTANT parameter is set to FIXED. For example, to pass a four bit value of 3 to the dataa[] port, the INPUT_A_FIXED_ VALUE parameter must be set to B0011.
INPUT_B_FIXED_VALUE	String	No	Specifies the value for the datab[] port. This parameter is used when the INPUT_B_ IS_CONSTANT parameter is set to FIXED. For example, to pass a four bit value of 3 to the datab[] port, the INPUT_B_FIXED_ VALUE parameter must be set to B0011.

ALTECC (Error Correction Code: Encoder/Decoder)

5

2014.12.19

UG-01063

The error correction code (ECC) is a error detection and correction method in digital data transmission. Its primary purpose is to detect corrupted data that occurs at the receiver side during data transmission. This error correction method is best suited for situations where errors occur at random rather than in bursts.

The ECC detects errors through the process of data encoding and decoding. For example, when the ECC is applied in a transmission application, data read from the source are encoded before being sent to the receiver. The output (code word) from the encoder consists of the raw data appended with the number of parity bits. The exact number of parity bits appended depends on the number of bits in the input data. The generated code word is then transmitted to the destination.

The receiver receives the code word and decodes it. Information obtained by the decoder determines whether an error is detected. The decoder detects single-bit and double-bit errors, but can only fix single-bit errors in the corrupted data. This type of ECC is called a single error correction double error detection (SECDED).

Altera provides two megafunctions, the ALTECC_ENCODER and ALTECC_DECODER, to implement the ECC functionality. The data input to the ALTECC_ENCODER megafunction is encoded to generate a code word that is a combination of the data input and the generated parity bits. The generated code word is transmitted to the ALTECC_DECODER megafunction for decoding just before reaching its destination block. The ALTECC_DECODER megafunction generates a syndrome vector to determine if there is any error in the received code word. It fixes the data only if the single-bit error is from the data bits. No signal is flagged if the single-bit error is from the parity bits. The megafunction also has flag signals to show the status of the data received and the action taken by the ALTECC_DECODER megafunction, if any.

The following figures show the ports for the ALTECC megafunction.

Figure 5-1: ALTECC_ENCODER Ports

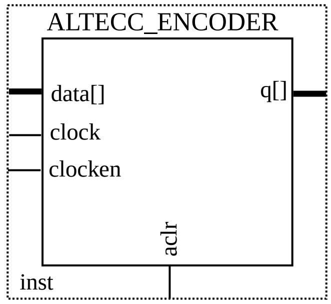
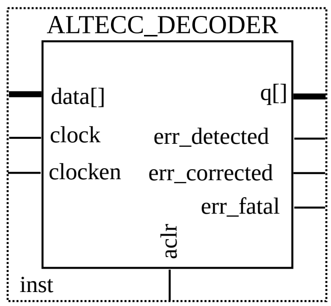


Figure 5-2: ALTECC_DECODER Ports



ALTECC_ENCODER Features

The ALTECC_ENCODER megafunction offers the following features:

- Performs data encoding using the Hamming Coding scheme
- Supports data width of 2–64 bits
- Supports signed and unsigned data representation format
- Support pipelining with output latency of either one or two clock cycles
- Supports optional asynchronous clear and clock enable ports

The ALTECC_ENCODER megafunction takes in and encodes the data using the Hamming Coding scheme. The Hamming Coding scheme derives the parity bits and appends them to the original data to produce the output code word. The number of parity bits appended depends on the width of the data.

The following table lists the number of parity bits appended for different ranges of data widths. The **Total Bits** column represents the total number of input data bits and appended parity bits.

Table 5-1: Number of Parity Bits and Code Word According to Data Width

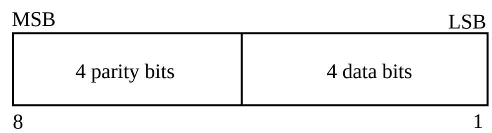
Data Width	Number of Parity Bits	Total Bits (Code Word)
2-4	3+1	6-8
5-11	4+1	10-16
12-26	5+1	18-32

Data Width	Number of Parity Bits	Total Bits (Code Word)
27-57	6+1	34-64
58-64	7+1	66-72

The parity bit derivation uses an even-parity checking. The additional 1 bit (shown in tthe table as +1) is appended to the parity bits as the MSB of the code word. This ensures that the code word has an even number of 1's. For example, if the data width is 4 bits, 4 parity bits are appended to the data to become a code word with a total of 8 bits. If 7 bits from the LSB of the 8-bit code word have an odd number of 1's, the 8th bit (MSB) of the code word is 1 making the total number of 1's in the code word even.

The following figure shows the generated code word and the arrangement of the parity bits and data bits in an 8-bit data input.

Figure 5-3: Parity Bits and Data Bits Arrangement in an 8-Bit Generated Code Word



The ALTECC_ENCODER megafunction accepts only input widths of 2 to 64 bits at one time. Input widths of 12 bits, 29 bits, and 64 bits, which are ideally suited to Altera devices, generate outputs of 18 bits, 36 bits, and 72 bits respectively. The bit-selection limitation is controlled by the MegaWizard Plug-In Manager.

Resource Utilization and Performance

The following tables provide resource utilization and performance information for the ALTECC megafunction.

Table 5-2: ALTECC Resource Utilization and Performance for Stratix III Devices

Configuration	Input data width	Output latency	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
ALTECC_ ENCODER	12	0	8	0	4	1161
	29	0	21	0	13	1076
	32	0	19	0	12	979
	64	0	40	0	27	758
	12	2	8	30	19	1188
	29	2	20	65	36	1021
	32	2	19	71	40	1013
	64	2	39	136	79	926
ALTECC_ DECODER	12	0	8	0	4	1161
	29	0	21	0	13	1076
	32	0	19	0	12	979
	64	0	40	0	27	758
	12	2	8	30	19	1188
	29	2	20	65	36	1021
	32	2	19	71	40	1013
	64	2	39	136	79	926

(3) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

Configuration	Input data width	Output latency	Logic Usage			f_{MAX} (MHz)
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
ALTECC_ ENCODER	12	0	8	0	4	1128
	29	0	21	0	13	1072
	32	0	19	0	12	1054
	64	0	40	0	27	901
	12	2	8	30	19	1104
	29	2	20	65	38	1082
	32	2	19	71	42	1061
	64	2	39	136	78	905
ALTECC_ DECODER	12	0	8	0	4	1128
	29	0	21	0	13	1072
	32	0	19	0	12	1054
	64	0	40	0	27	901
	12	2	8	30	19	1104
	29	2	20	65	38	1082
	32	2	19	71	42	1061
	64	2	39	136	78	905

Verilog HDL Prototype (ALTECC_ENCODER)

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **lpm.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module altecc_encoder
#( parameter intended_device_family = "unused",
parameter lpm_pipeline = 0,
parameter width_codeword = 8,
parameter width_dataword = 8,
parameter lpm_type = "altecc_encoder",
parameter lpm_hint = "unused")
( input wire aclr,
input wire clock,
input wire clocken,
input wire [width_dataword-1:0] data,
output wire [width_codeword-1:0] q);
endmodule
```

Verilog HDL Prototype (ALTECC_DECODER)

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **lpm.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module altecc_decoder
#( parameter intended_device_family = "unused",
parameter lpm_pipeline = 0,
parameter width_codeword = 8,
parameter width_dataword = 8,
parameter lpm_type = "altecc_decoder",
parameter lpm_hint = "unused")
( input wire aclr,
input wire clock,
input wire clocken,
input wire [width_codeword-1:0] data,
output wire err_corrected,
output wire err_detected,
outut wire err_fatal,
output wire [width_dataword-1:0] q);
endmodule
```

VHDL Component Declaration (ALTECC_ENCODER)

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altecc_encoder
generic (
intended_device_family:string := "unused";
lpm_pipeline:natural := 0;
width_codeword:natural := 8;
width_dataword:natural := 8;
lpm_hint:string := "UNUSED";
lpm_type:string := "altecc_encoder");
port(
aclr:in std_logic := '0';
clock:in std_logic := '0';
clocken:in std_logic := '1';
data:in std_logic_vector(width_dataword-1 downto 0);
q:out std_logic_vector(width_codeword-1 downto 0));
end component;
```

VHDL Component Declaration (ALTECC_DECODER)

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altecc_decoder
generic (
intended_device_family:string := "unused";
lpm_pipeline:natural := 0;
width_codeword:natural := 8;
width_dataword:natural := 8;
lpm_hint:string := "UNUSED";
lpm_type:string := "altecc_decoder");
port(
aclr:in std_logic := '0';
clock:in std_logic := '0';
clocken:in std_logic := '1';
```

```
data:in std_logic_vector(width_codeword-1 downto 0);
q:out std_logic_vector(width_dataword-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

Ports (ALTECC_ENCODER)

The following tables list the input and output ports for the ALTECC_ENCODER megafunction.

Table 5-3: ALTECC_ENCODER Megafunction Input Ports

Port Name	Required	Description
`data[]`	Yes	Data input port. The size of the input port depends on the `WIDTH_DATAWORD` parameter value. The data[] port contains the raw data to be encoded.
`clock`	Yes	Clock input port that provides the clock signal to synchronize the encoding operation. The clock port is required when the `LPM_PIPELINE` value is greater than 0.
`clocken`	No	Clock enable. If omitted, the default value is `1`.
`aclr`	No	Asynchronous clear input. The active high `aclr` signal can be used at any time to asynchronously clear the registers.

Table 5-4: ALTECC_ENCODER Megafunction Output Ports

Port Name	Required	Description
`q[]`	Yes	Encoded data output port. The size of the output port depends on the `WIDTH_CODEWORD` parameter value.

Ports (ALTECC_DECODER)

The following tables list the input and output ports for the ALTECC_DECODER megafunction.

Table 5-5: ALTECC_DECODER Megafunction Input Ports

Port Name	Required	Description
`data[]`	Yes	Data input port. The size of the input port depends on the `WIDTH_CODEWORD` parameter value.

Port Name	Required	Description
clock	Yes	Clock input port that provides the clock signal to synchronize the encoding operation. The clock port is required when the LPM_PIPELINE value is greater than 0.
clocken	No	Clock enable. If omitted, the default value is 1.
aclr	No	Asynchronous clear input. The active high aclr signal can be used at any time to asynchronously clear the registers.

Table 5-6: ALTECC_DECODER Megafunction Output Ports

Port Name	Required	Description
q[]	Yes	Decoded data output port. The size of the output port depends on the WIDTH_DATAWORD parameter value.
err_detected	Yes	Flag signal to reflect the status of data received and specifies any errors found.
err_corrected	Yes	Flag signal to reflect the status of data received. Denotes single-bit error found and corrected. You can use the data because it has already been corrected.
err_fatal	Yes	Flag signal to reflect the status of data received. Denotes double-bit error found, but not corrected. You must not use the data if this signal is asserted.
syn_e	No	An output signal which will go high whenever a single-bit error is detected on the parity bits.

Parameters (ALTECC_ENCODER)

The following table lists the parameters for the ALTECC_ENCODER megafunction.

Table 5-7: ALTECC_ENCODER Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_DATAWORD	Integer	Yes	Specifies the width of the raw data. Values are from 2 to 64. If omitted, the default value is 8.
WIDTH_CODEWORD	Integer	Yes	Specifies the width of the corresponding code word. Valid values are from 6 to 72, excluding 9, 17, 33, and 65. If omitted, the default value is 13.
LPM_PIPELINE	Integer	No	Specifies the pipeline for the circuit. Values are from 0 to 2. If the value is 0, the ports are not registered. If the value is 1, the output ports are registered. If the value is 2, the input and output ports are registered. If omitted, the default value is 0.

Parameters (ALTECC_DECODER)

The following table lists the parameters for the ALTECC_DECODER megafunction.

Table 5-8: ALTECC_DECODER Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_DATAWORD	Integer	Yes	Specifies the width of the raw data. Values are 2 to 64. The default value is 8.
WIDTH_CODEWORD	Integer	Yes	Specifies the width of the corresponding code word. Values are 6 to 72, excluding 9, 17, 33, and 65. If omitted, the default value is 13.
LPM_PIPELINE	Integer	No	Specifies the register of the circuit. Values are from 0 to 2. If the value is 0, no register is implemented. If the value is 1, the output is registered. If the value is 2, both the input and the output are registered. If the value is greater than 2, additional registers are implemented at the output for the additional latencies. If omitted, the default value is 0.
Create a 'syn_e' port	Integer	No	Turn on this parameter to create a syn_e port.

Design Example 1: ALTECC_ENCODER

This design example uses the ECC encoder to encode an 8-bit wide input data to generate 13 bits of output code word. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in altecc_DesignExample1.zip:

- **altecc_encode.qar** (archived Quartus II design files)
- **altecc_encode_ex_msim** (ModelSim-Altera files)

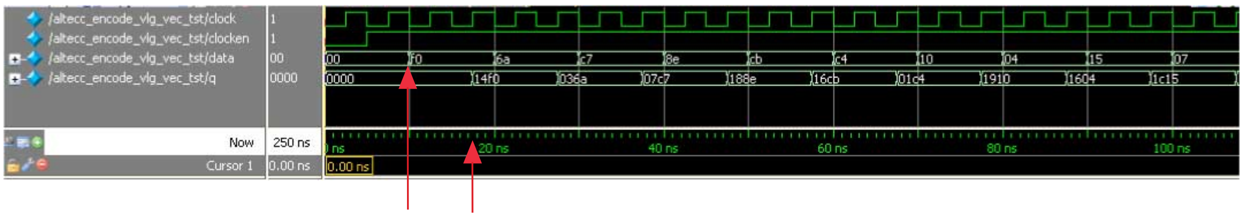
Understanding the Simulation Results

The following settings are observed in this example:

- The data[] input width is set to 8 bits
- The output port, q[] has a width of 13 bits
- The clock enable (clocken) signal is enabled
- Pipelining is enabled, with an output latency of 2 clock cycles. Hence, the result is seen on the q[] port two clock cycles after the input data is available

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 5-4: Design Example 1: Simulation Waveform for the ECC Encoder



The following sequence corresponds with the numbered items in the figure:

- Data F0 is fed to the ECC encoder. As pipelining is enabled to have an output latency of 2 clock cycles, the result of the encoding operation appears at the output port q[] 2 clock cycles later. The 8-bit input data (F0) is encoded to generate a 13-bit output code word (14F0). The input data is appended with 5 parity bits. The ECC encoder encodes the data based on the Hamming Code scheme. The following steps describe the Hamming Code algorithm and explain how the ECC encoder encodes input data F0 to generate the output code word of 14F0:
    - In a 13-bit code word, there are 13 locations (bit positions), and each location holds 1 bit. There are 8 bits of original data, and the appended 5 parity bits. The locations (bit positions) for the bits must be defined – bit positions that are powers of 2 are used as parity bits (positions 1, 2, 4, 8 …).
    - The following table lists the bit positions, and the position of the parity bits of a 13-bit code word. P5* is the extra parity bit added. The prefix P denotes parity.

Table 5-9: Design Example 1: Position of Parity Bits for a 13-Bit Code Word

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Parity Bits	P1	P2	—	P3	—	—	—	P4	—	—	—	—	P5*

- All other bit positions are for the data to be encoded. The least significant bit (LSB) of the data bit fills the lowest bit position. In this case, starting from the LSB of the data, F0 (1111 0000 in binary) fills the empty bit positions, starting from position (3), as shown in the following table. The prefixes P and D denote parity and data, respectively. For the standard Hamming Code algorithm, the MSB of the data bit fills the lowest bit position, unlike the Altera ECC megafunction, which fills up the lowest bit position starting with the LSB. This bit order reduces the complexity of the circuit design.

Table 5-10: Design Example 1: Filling of Data Bits (1111 0000) for a 13-Bit Code Word

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Parity Bits and Data Bits	P1	P2	D1 0	P3	D2 0	D3 0	D4 0	P4	D5 1	D6 1	D7 1	D8 1	P5*

- Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.

  The following section list the sequence of bits that each parity bit checks:

  Parity bit 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit… (1, 3, 5, 7, 9, 11)

  Parity bit 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits… (2, 3, 6, 7, 10, 11)

  Parity bit 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits… (4, 5, 6, 7,1 2)

  Parity bit 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits… (8, 9, 10, 11, 12)

Table 5-11: Design Example 1: Calculation of Parity Bits

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Parity Bits and Data Bits	P1	P2	D1 0	P3	D2 0	D3 0	D4 0	P4	D5 1	D6 1	D7 1	D8 1	P5*
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	—	—
Calculate P2	—	0	0	—	—	0	0	—	—	1	1	—	—
Calculate P3	—	—	—	1	0	0	0	—	—	—	—	1	—
Calculate P4	—	—	—	—	—	—	—	0	1	1	1	1	—
Calculate P5	0	0	0	1	0	0	0	0	1	1	1	1	1

- The encoded input data for F0 is 14F0 (1 0100 1111 0000 in binary), as seen on the output port, q[], at 17.5 ns.

Design Example 1: Calculation of Parity Bits

Design Example 2: ALTECC_DECODER

This design example uses the ECC decoder to decode input code words of 13-bit widths to generate 8 bits of output data. An asynchronous clear signal is also used to illustrate how the signal affects the registered ports. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **altecc_DesignExample2.zip**:

- **altecc_decode.qar** (archived Quartus II design files)
- **altecc_decode_ex_msim** (ModelSim-Altera files)

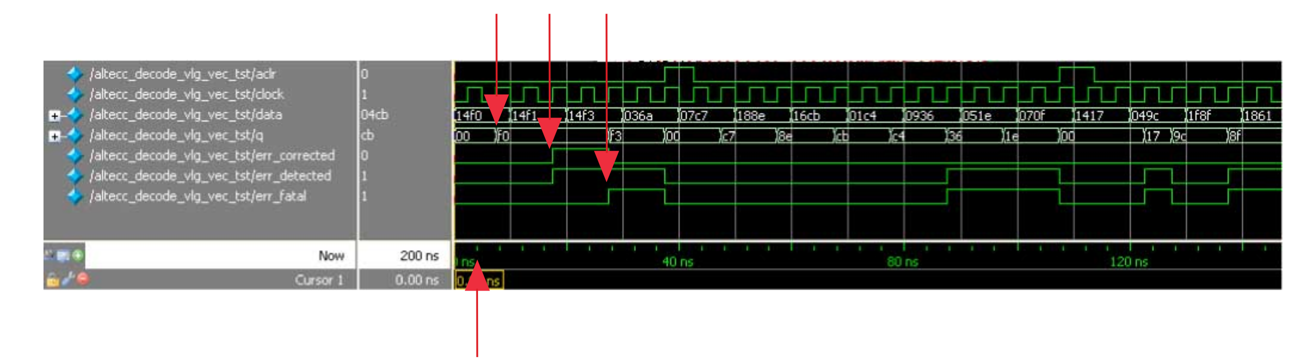
Understanding the Simulation Results

The following settings are observed in this example:

- The `data[]` input width is set to 13 bits
- The output port, `q[]` has a width of 8 bits
- The asynchronous clear (`aclr`) signal is enabled
- Pipelining is enabled, with an output latency of 2 clock cycles. Hence, the result is seen on the `q[]` port two clock cycles after the input data is available

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 5-6: Design Example 2: Simulation Waveform for the ECC Decoder



The following sequence corresponds with the numbered items in the figure.

1. The decoder decodes the code word 14F0 at the first rising edge of the clock at 2.5 ns. In this case, the input code word is not corrupted. The 13-bit input code word 14F0 (1 0100 1111 0000 in binary) is decoded to generate an 8-bit output data of F0. The following table lists the arrangement of parity bits and data bits in the code word 14F0. The prefixes P and D denote parity and data respectively.

Table 5-12: Design Example 2: Arrangement of Parity Bits and Data Bits in Code Word 14F0

MSB												LSB
P5*	P4	P3	P2	P1	D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	0	1	1	1	1	0	0	0	0

The ECC decoder decodes the code word based on the Hamming Code scheme. The following steps describe the Hamming Code algorithm and explain how the ECC decoder decodes input code word 14F0 to generate output data F0:

- All bits have their bit positions, and bit positions that are powers of 2 are used as parity bits (positions 1, 2, 4, 8 …). Table 38 lists the bit positions and the positions of the parity bits in a 13-bit code word.

Table 5-13: Design Example 2: Position of Parity Bits for a 13-Bit Code Word

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Parity Bits and Data Bits	P1 0	P2 0	—	P3 1	—	—	—	P4 0	—	—	—	—	P5 1

- All other bit positions are for the data bits. The LSB of the data bit fills the lowest bit position. In this case, starting from the LSB of the data, F0 (1111 0000 in binary) fills the empty bit positions, starting from position (3), as shown in the following table.

Table 5-14: Design Example 2: Filling of Data Bits (1111 0000) for a 13-Bit Code Word

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Parity Bits and Data Bits	P1 0	P2 0	D1 0	P3 1	D2 0	D3 0	D4 0	P4 0	D5 1	D6 1	D7 1	D8 1	P5 1

- Recalculate parity bits to generate the syndrome code. Each syndrome bit calculates the parity (even parity) for some of the bits in the code word. The following table lists how the syndrome bits are derived.

Table 5-15: Design Example 2: Calculation of Parity Bits

Position	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	
Parity Bits and Data Bits	P1 0	P2 0	D1 0	P3 1	D2 0	D3 0	D4 0	P4 0	D5 1	D6 1	D7 1	D8 1	P5 1	Syndrome Code
Calculate P1	0	—	0	—	0	—	0	—	1	—	1	—	—	S1=0
Calculate P2	—	0	0	—	—	0	0	—	—	1	1	—	—	S2=0
Calculate P3	—	—	—	1	0	0	0	—	—	—	—	1	—	S3=0
Calculate P4	—	—	—	—	—	—	—	0	1	1	1	1	—	S4=0
Calculate P5	0	0	0	1	0	0	0	0	1	1	1	1	1	S5*=0

- Calculate the additional syndrome bit using an even parity checking on all the bits in the code word. In this example, the additional syndrome bit S5* is calculated using an even parity checking on all the bits from position (1) to position (13) as shown in the table. The generated syndrome code gives the status of the data, whether an error has occurred, and if so, whether it is a single-bit or double-bit error.

2. In this case, the syndrome code is zero (S5*S4S3S2S1=0 0000). No error is detected and no correction

is needed on the retrieved data F0 (D8D7D6D5D4D3D2D1=1111 0000) based on the generated syndrome code. Therefore, the flag signals `err_detected`, `err_corrected`, and `err_fatal` are deasserted, indicating that the data is not corrupted. The decoding for 14F0 is F0 (1111 0000 in binary).

Note: Even if the generated syndrome code indicates a single-bit error, the `err_detected` and `err_corrected` signals are asserted only if the corrupted bit is from the data bits and not from the parity bits.

3. At 10 ns, a single-bit error occurred in the input code word that changes the code word to 14F1. In this case, assume that one of the data bits, the LSB, is corrupted and is inverted from 0 to 1. This causes the code word to become 14F1.

   With the same method of decoding using the Hamming Code scheme, the generated syndrome code is 1 0011. S5* equals to 1 (single error detected), and S4S3S2S1 equals to 0011 (the bit at position 3 is corrupted).

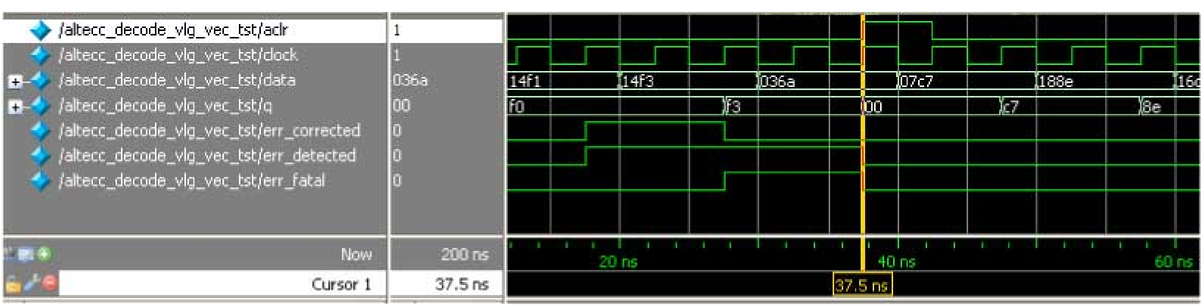
   Because only one of the data bits is corrupted, the decoder is able to correct it by flipping the error bit. Therefore, the corrupted data F1 is decoded as F0. When F0 is shown at the output port at the next rising edge of the clock at 17.5 ns, the `err_detected` and `err_corrected` signals are asserted to show that an error is detected and the single-bit error is corrected.
4. At 20 ns, a double-bit error in the input code word changes the code word to 14F3.

   In this case, assume that two of the data bits (bit-0 and bit-1) are corrupted and are inverted from 0 to 1. This causes the code word to become 14F3.

   The decoder decodes the code word 14F3 at 20 ns and shows the data F3 at 27.5 ns. The ECC decoder performs only SECDED, therefore it does not fix the corrupted data that contains double-bit errors. Instead, the `err_fatal` signal is asserted together with the `err_detected` signal.

   The following figure shows the effects of the asynchronous-clear signal on the registered ports.

Figure 5-7: Design Example 2: Asynchronous-Clear Feature of ECC



This figure shows that when the `aclr` signal is asserted at 37.5 ns, the output and status signals are cleared immediately.

If you do not want to use the corrupted data when the `err_fatal` signal is asserted, you can assert the asynchronous-clear signal (`aclr`) to clear the output port q and other status signals that are registered. You must enable the pipelining option in the MegaWizard Plug-In Manager to use this feature.

ALTERA_MULT_ADD (Multiply-Adder) 6

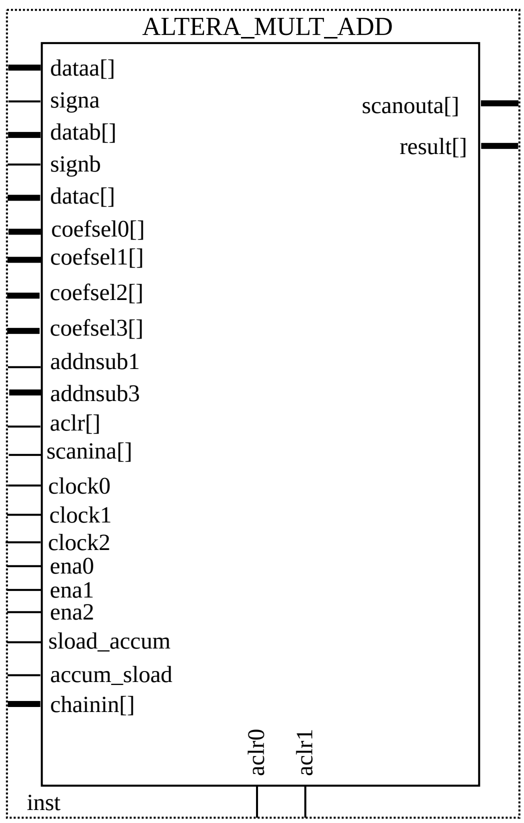
2014.12.19

UG-01063 Subscribe Send Feedback

The ALTERA_MULT_ADD megafunction allows you to implement a multiplier-adder.

The following figure shows the ports for the ALTERA_MULT_ADD megafunction.

Figure 6-1: ALTERA_MULT_ADD Ports



A multiplier-adder accepts pairs of inputs, multiplies the values together and then adds to or subtracts from the products of all other pairs.

The ALTERA_MULT_ADD megafunction also offers many variations in dedicated DSP block circuitry. Data input sizes of up to 18 bits are accepted. Because the DSP blocks allow for one or two levels of 2-input add or subtract operations on the product, this function creates up to four multipliers.

For Stratix V devices, the multiplier blocks and adder/accumulator block is combined in a single MAC block.

The multipliers and adders of the ALTERA_MULT_ADD megafunction are placed in the dedicated DSP block circuitry of the Stratix devices. If all of the input data widths are 9-bits wide or smaller, the function uses the 9 × 9-bit input multiplier configuration in the DSP block. If not, the DSP block uses 18 × 18-bit input multipliers to process data with widths between 10 bits and 18 bits. If multiple ALTERA_MULT_ADD megafunctions occur in a design, the functions are distributed to as many different DSP blocks as possible so that routing to these blocks is more flexible. Fewer multipliers per DSP block allow more routing choices into the block by minimizing paths to the rest of the device.

The registers and extra pipeline registers for the following signals are also placed inside the DSP block:

- Data input
- Signed or unsigned select
- Add or subtract select
- Products of multipliers

In the case of the output result, the first register is placed in the DSP block. However the extra latency registers are placed in logic elements outside the block. Peripheral to the DSP block, including data inputs to the multiplier, control signal inputs, and outputs of the adder, use regular routing to communicate with the rest of the device. All connections in the function use dedicated routing inside the DSP block. This dedicated routing includes the shift register chains when you select the option to shift a multiplier's registered input data from one multiplier to an adjacent multiplier.

For more information about DSP blocks in any of the Stratix, Stratix GX, and Arria GX device series, refer to the DSP Blocks chapter of the respective handbooks on the **Literature and Technical Documentation** page.

For more information about the embedded memory blocks in any of the Stratix, Stratix GX, and Arria GX device series, refer to the *TriMatrix Embedded Memory Blocks* chapter of the respective handbooks on the **Literature and Technical Documentation** page.

For more information on embedded multiplier blocks in the Cyclone II and Cyclone III devices, refer to the *DSP Blocks* chapter of the respective handbooks on the **Literature and Technical Documentation** page.

For more information about implementing multipliers using DSP and memory blocks in Altera FPGAs, refer to **AN 306: Implementing Multipliers in FPGA Devices**.

Features

The ALTERA_MULT_ADD megafunction offers the following features:

- Generates a multiplier to perform multiplication operations of two complex numbers

  Note: When building multipliers larger than the natively supported size there may/will be a performance impact resulting from the cascading of the DSP blocks.
- Supports data widths of 1– 256 bits
- Supports signed and unsigned data representation format
- Supports pipelining with configurable output latency
- Provides an option to dynamically switch between signed and unsigned data support

- Provides an option to dynamically switch between add and subtract operation
- Supports optional asynchronous clear and clock enable input ports
- Supports systolic delay register mode
- Supports pre-adder with 8 pre-load coefficients per multiplier
- Supports pre-load constant to complement accumulator feedback
- Pre-adder, coefficient storage and systolic delay register features are added to maximize flexibility.

Pre-adder

With pre-adder, additions or subtractions are done prior to feeding the multiplier.

There are five pre-adder modes:

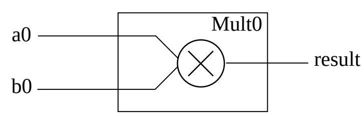
- Simple mode
- Coefficient mode
- Input mode
- Square mode
- Constant mode

Note: When pre-adder is used (pre-adder coefficient/input/square mode), all data inputs to the multiplier must have the same clock setting.

Pre-adder Simple Mode

In this mode, both operands derive from the input ports and pre-adder is not used or bypassed. This is the default mode.

Figure 6-2: Pre-adder Simple Mode



Pre-adder Coefficient Mode

In this mode, one multiplier operand derives from the pre-adder, and the other operand derives from the internal coefficient storage. The coefficient storage allows up to 8 preset constants. The coefficient selection signals are `coefsel[0..3]`.

The following settings are applied in this mode:

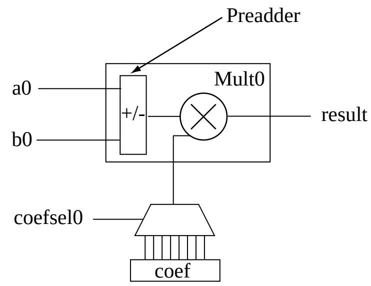
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 25 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 25 bits
- The width of the coefficient input must be less than or equals to 27 bits

This mode is expressed in the following equation.

$$y = (a + b) \times coef$$

The following shows the pre-adder coefficient mode of a multiplier.

Figure 6-3: Pre-adder Coefficient Mode



Pre-adder Input Mode

In this mode, one multiplier operand derives from the pre-adder, and the other operand derives from the `datac[]` input port.

The following settings are applied in this mode:

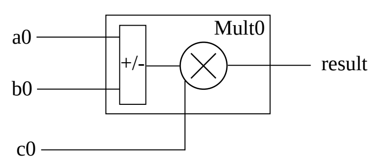
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 25 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 25 bits
- The width of the `datac[]` input (`WIDTH_C`) must be less than or equals to 22 bits
- The number of multipliers must be set to 1
- All input registers must be registered with the same clock

This mode is expressed in the following equation.

$$y = (a + b) \times c$$

The following shows the pre-adder input mode of a multiplier.

Figure 6-4: Pre-adder Input Mode



Pre-adder Square Mode

In this mode, both multiplier operands derive from the pre-adder.

The following settings are applied in this mode:

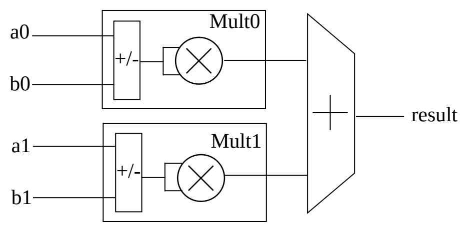
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 17 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 17 bits
- The number of multipliers must be set to 2

This mode is expressed in the following equation.

$$y = (a0 + b0)^2 + (a1 + b1)^2$$

The following shows the pre-adder square mode of two multipliers.

Figure 6-5: Pre-adder Square Mode



Pre-adder Constant Mode

In this mode, one multiplier operand derives from the input port, and the other operand derives from the internal coefficient storage. The coefficient storage allows up to 8 preset constants. The coefficient selection signals are `coefsel[0..3]`.

The following settings are applied in this mode:

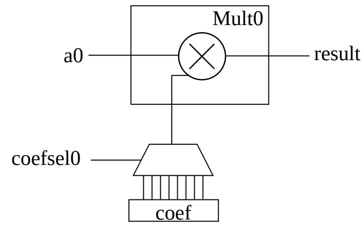
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 27 bits
- The width of the coefficient input must be less than or equals to 27 bits
- The `datab[]` port must be disconnected

This mode is expressed in the following equation.

$$y = a0 \times coef$$

The following figure shows the pre-adder constant mode of a multiplier.

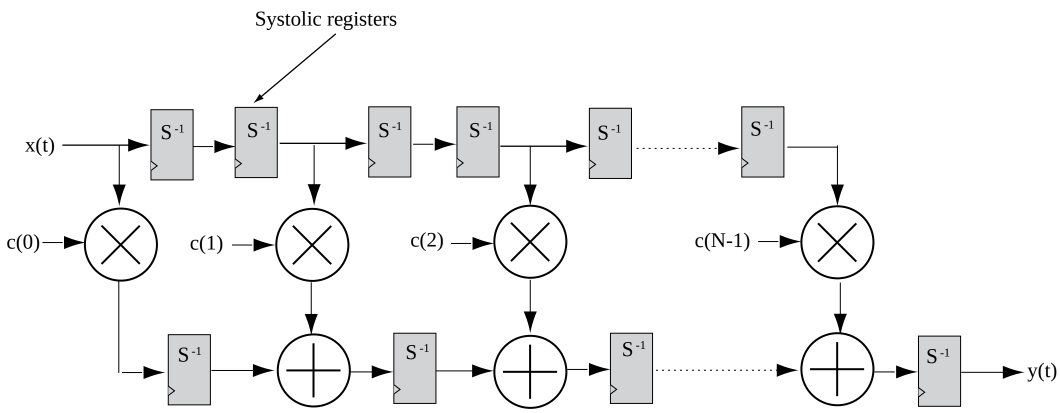
Figure 6-6: Pre-adder Constant Mode



Systolic Delay Register

In a systolic architecture, the input data is fed into a cascade of registers acting as a data buffer. Each register delivers an input sample to a multiplier where it is multiplied by the respective coefficient. The chain adder stores the gradually combined results from the multiplier and the previously registered result from the `chainin[]` input port to form the final result. Each multiply-add element must be delayed by a single cycle so that the results synchronize appropriately when added together. Each successive delay is used to address both the coefficient memory and the data buffer of their respective multiply-add elements. For example, a single delay for the second multiply add element, two delays for the third multiply-add element, and so on.

Figure 6-7: Systolic Registers



x(t) represents the results from a continuous stream of input samples and *y(t)* represents the summation of a set of input samples, and in time, multiplied by their respective coefficients. Both the input and output results flow from left to right. The *c(0)* to *c(N*-1) denotes the coefficients. The systolic delay registers are denoted by S^{-1}, whereas the $^{-1}$ represents a single clock delay. Systolic delay registers are added at the inputs and outputs for pipelining in a way that ensures the results from the multiplier operand and the accumulated sums stay in synch. This processing element is replicated to form a circuit that computes the filtering function. This function is expressed in the following equation.

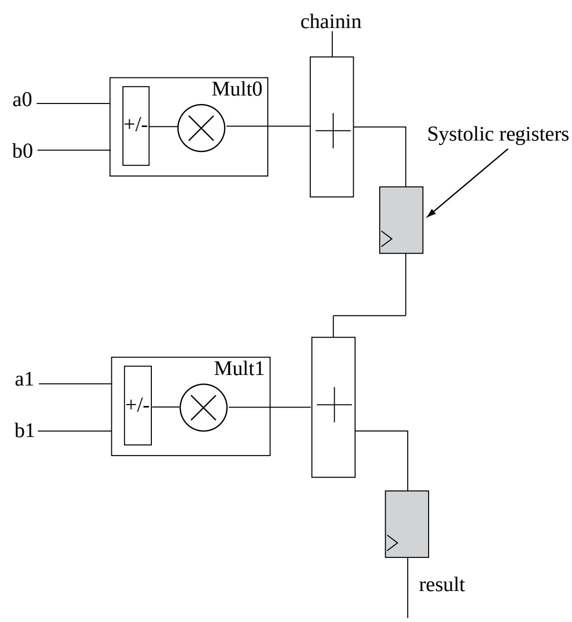
$$y(t) = \sum_{i=0}^{N-1} B(i)A(t-i)$$

N represents the number of cycles of data that has entered into the accumulator, *y(t)* represents the output at time *t*, *A(t)* represents the input at time *t*, and *B(i)* are the coefficients. The *t* and *i* in the equation correspond to a particular instant in time, so to compute the output sample *y(t)* at time t, a group of input samples at *N* different points in time, or A(n), A(n-1), A(n-2), … A(n-N+1) is required. The group of N input samples are multiplied by *N* coefficients and summed together to form the final result *y*.

The systolic register architecture is available only for sum-of-2 and sum-of-4 modes.

The following figure shows the systolic delay register implementation of 2 multipliers.

Figure 6-8: Systolic Delay Register Implementation of 2 Multipliers

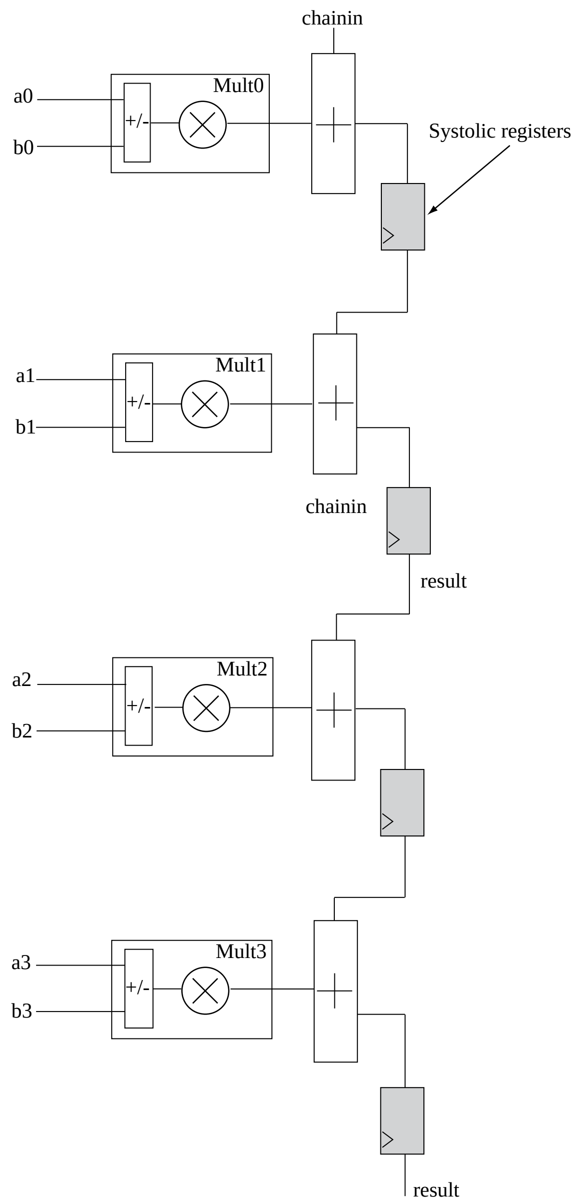


The sum of two multipliers is expressed in the following equation.

$$y(t) = [a1(t) \times b1(t)] + [a0(t-1) \times b0(t-1)]$$

The following figure shows the systolic delay register implementation of 4 multipliers.

Figure 6-9: Systolic Delay Register Implementation of 4 Multipliers



The sum of four multipliers is expressed in the following equation.

Figure 6-10: Sum of 4 Multipliers

$$y(t) = [a3(t) \times b3(t)] + [a2(t-1) \times b2(t-1)] + [a1(t-2) \times b1(t-2)] + [a0(t-3) \times b0(t-3)]$$

The following lists the advantages of systolic register implementation:

- Reduces DSP resource usage
- Enables efficient mapping in the DSP block using the chain adder structure

The systolic delay implementation is only available for the following pre-adder modes:

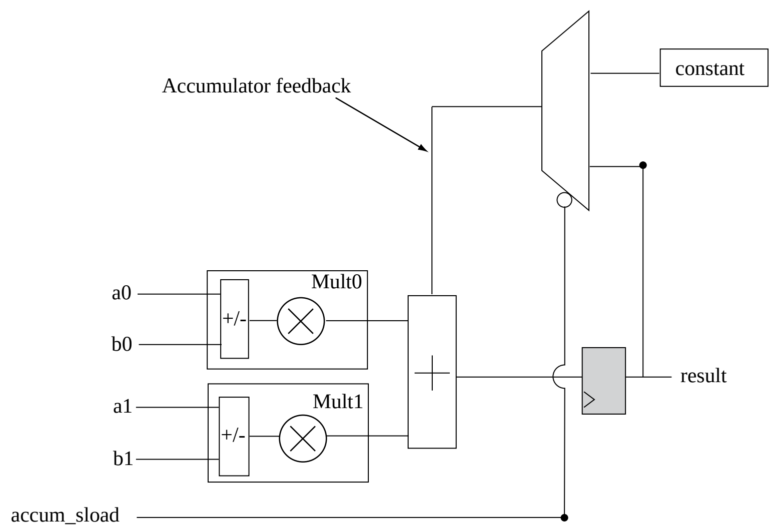
- Pre-adder coefficient mode
- Pre-adder simple mode
- Pre-adder constant mode

Pre-load Constant

The pre-load constant controls the accumulator operand and complements the accumulator feedback. The valid `LOADCONST_VALUE` ranges from 0–64. The constant value is equal to 2^N, where N = `LOADCONST_VALUE`. When the `LOADCONST_VALUE` is set to 64, the constant value is equal to 0. This function can be used as biased rounding.

The following figure shows the pre-load constant implementation.

Figure 6-11: Pre-load Constant



Refer to the following megafunctions in this user guide for other multiplier implementations:

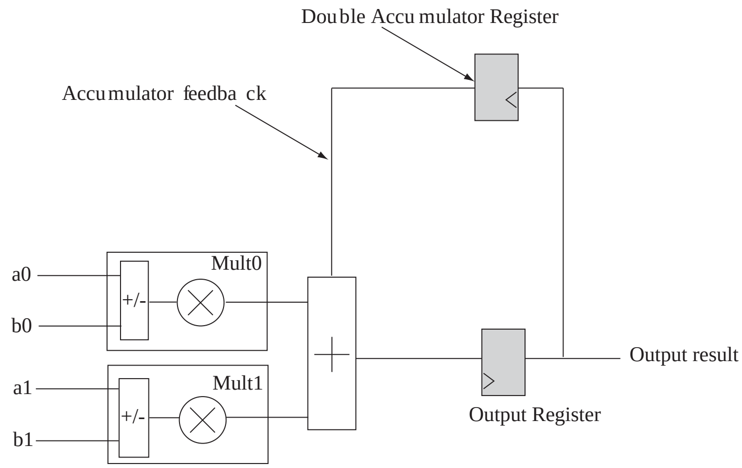
- **ALTMULT_ACCUM (Multiply-Accumulate)**
- **ALTMEMMULT (Memory-based Constant Coefficient Multiplier)**
- **LPM_MULT (Multiplier)**

Double Accumulator

The double accumulator feature adds an additional register in the accumulator feedback path. The double accumulator register follows the output register, which includes the clock, clock enable, and aclr. The additional accumulator register returns result with a one-cycle delay. This feature enables you to have two accumulator channels with the same resource count.

The following figure shows the double accumulator implementation.

Figure 6-12: Double Accumulator



Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) in the <*Quartus II installation directory*>**\eda\synthesis** directory.

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) in the <*Quartus II installation directory*> directory.

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

Ports

The following tables list the input and output ports of the ALTERA_MULT_ADD megafunction.

Table 6-1: ALTERA_MULT_ADD MegaFunction Input Ports

Port name	Required	Description
`dataa []`	Yes	Data input to the multiplier. Input port `[NUMBER_OF_ MULTIPLIERS * WIDTH_A - 1 … 0]` wide

Port name	Required	Description
datab []	Yes	Data input to the multiplier. Input port [NUMBER_OF_ MULTIPLIERS * WIDTH_B - 1 … 0] wide
datac []	No	Data input to the multiplier. Input port [NUMBER_OF_ MULTIPLIERS * WIDTH_C - 1 … 0] wide
clock []	No	Clock input port [0 … 2] to the corresponding register. This port can be used by any register in the megafunction.
aclr []	No	Input port [0 ... 1]. Asynchronous clear input to the corresponding register.
ena []	No	Input port [0 ... 2]. Enable signal input to the corresponding register.
signa	No	Specifies the numerical representation of the multiplier input A. If the signa port is high, the multiplier treats the multiplier input A port as a signed number. If the signa port is low, the multiplier treats the multiplier input A port as an unsigned number.
signb	No	Specifies the numerical representation of the multiplier input B port. If the signb port is high, the multiplier treats the multiplier input B port as a signed two's complement number. If the signb port is low, the multiplier treats the multiplier input B port as an unsigned number.
scanina[]	No	Input for scan chain A. Input port [WIDTH_A - 1 ... 0] wide. When the INPUT_SOURCE_A parameter has a value of SCANA, the scanina[] port is required.
accum_sload	No	Dynamically specifies whether the accumulator value is constant. If the accum_sload port is high, then the multiplier output is loaded into the accumulator. Do not use accum_sload and sload_accum simultaneously.
sload_accum	No	Dynamically specifies whether the accumulator value is constant. If the sload_accum port is low, then the multiplier output is loaded into the accumulator. Do not use accum_sload and sload_accum simultaneously.
chainin []	No	Adder result input bus from the preceding stage. Input port [WIDTH_CHAININ - 1 … 0] wide.
addnsub1	No	Controls the functionality of the first adder. If the addnsub1 port is high, the first adder performs an add function. If the addnsub1 port is low, the adder performs a subtract function.
addnsub3	No	Controls the functionality of the first adder. If the addnsub3 port is high, the first adder performs an add function. If the addnsub3 port is low, the adder performs a subtract function.
coefsel0 []	No	Coefficient input port[0..3] to the first multiplier.
coefsel1 []	No	Coefficient input port[0..3]to the second multiplier.
coefsel2 []	No	Coefficient input port[0..3]to the third multiplier.
coefsel3 []	No	Coefficient input port [0..3] to the fourth multiplier.

Table 6-2: ALTERA_MULT_ADD MegaFunction Output Ports

Port Name	Required	Description
`result []`	Yes	Multiplier output port. Output port `[WIDTH_RESULT - 1 … 0]` wide
`scanouta []`	No	Output of scan chain A. Output port `[WIDTH_A - 1..0]` wide.

ALTERA_MULT_ADD Parameters

The following table lists the parameters for the ALTERA_MULT_ADD megafunction.

Table 6-3: ALTMULT_ADD Megafunction Parameters

Parameter Name	Type	Required	Description
`NUMBER_OF_MULTIPLIERS`	Integer	Yes	Number of multipliers to be added together. Values are `1` up to `4`.
`WIDTH_A`	Integer	Yes	Width of the `dataa[]` port.
`WIDTH_B`		Yes	Width of the `datab[]` port.
`WIDTH_RESULT`	Integer	Yes	Width of the `result[]` port.
`INPUT_REGISTER_A[0…3]`	String	No	Specifies the clock port for the `dataa[]` operand of the multiplier. Values are `UNREGISTERED`, `CLOCK0`, `CLOCK1`, and `CLOCK2`. If omitted, the default value is `UNREGISTERED`. `INPUT_REGISTER_A[1 … 3]` must have similar values with `INPUT_REGISTER_A0`.
`INPUT_REGISTER_B[0…3]`	String	No	Specifies the clock port for the `datab[]` operand of the multiplier. Values are `UNREGISTERED`, `CLOCK0`, `CLOCK1`, and `CLOCK2`. If omitted, the default value is `UNREGISTERED`. `INPUT_REGISTER_B[1 … 3]` must have similar values with `INPUT_REGISTER_B0`.
`INPUT_ACLR_A[0…3]`	String	No	Specifies the asynchronous clear for the `dataa[]` operand of the multiplier. Values are `NONE`, `ACLR0`, `ACLR1`. If omitted, the default value is `NONE`. The `INPUT_ACLR_A[1 … 3]` value must be set similar to the value of `INPUT_ACLR_A0`.
`INPUT_ACLR_B[0…3]`	String	No	Specifies the asynchronous clear for the `datab[]` operand of the multiplier. Values are `NONE`, `ACLR0`, `ACLR1`. If omitted, the default value is `NONE`. The `INPUT_ACLR_B [1 … 3]` value must be set similar to the value of `INPUT_ACLR_B0`.

Parameter Name	Type	Required	Description
INPUT_SOURCE_A[0…3]	String	No	Specifies the data source to the first multiplier. Values are DATAA and SCANA. If this parameter is set to DATAA, the adder uses the values from the dataa[] port. If this parameter is set to SCANA, the adder uses values from the scanina[]. If omitted, the default value is DATAA.
REPRESENTATION_A	String	No	Specifies the numerical representation of the multiplier input A. Values are UNSIGNED and SIGNED. When this parameter is set to SIGNED, the adder interprets the multiplier input A as a signed number. When this parameter is set to UNSIGNED, the adder interprets the multiplier input A as an unsigned number. If omitted, the default value is UNSIGNED. If the corresponding PORT_SIGNA value is USED, this parameter is ignored. Use the parameter PORT_SIGNA to access the signa input port for dynamic control of the representation through the signa input port.
REPRESENTATION_B	String	No	Specifies the numerical representation of the multiplier input B. Values are UNSIGNED and SIGNED. When this parameter is set to SIGNED, the adder interprets the multiplier input B as a signed number. When this parameter is set to UNSIGNED, the adder interprets the multiplier input B as an unsigned number. If omitted, the default value is UNSIGNED. If the corresponding PORT_SIGNB value is USED, this parameter is ignored. Use the parameter PORT_SIGNB to access the signb input port for dynamic control of the representation through the signb input port.
SIGNED_REGISTER_[]	String	No	Parameter [A, B]. Specifies the clock signal for the first register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If the corresponding sign[] port value is UNUSED, this parameter is ignored. If omitted, the default value is UNREGISTERED. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.

Parameter Name	Type	Required	Description
SIGNED_ACLR_[]	String	No	Parameter [A, B]. Specifies the asynchronous clear signal for the first register on the corresponding sign[] port. Values are NONE, ACLR0, and ACLR1. If omitted the default value is NONE. The value must be set similar to the value of INPUT_ACLR_A0.
MULTIPLIER_REGISTER[]	String	No	Parameter [0...3]. Specifies the clock source of the register that follows the corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGISTERED.
MULTIPLIER_ACLR[]	String	No	Parameter [0...3]. Specifies the asynchronous clear signal of the register that follows the corresponding multiplier. Values are NONE, ACLR0, and ACLR1. If omitted the default value is NONE.
MULTIPLIER1_DIRECTION	String	No	Specifies whether the second multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If the addnsub1 port is used, this parameter is ignored. If omitted, the default value is ADD.
MULTIPLIER3_DIRECTION	String	No	Specifies whether the fourth multiplier adds or subtracts their results from the total. Values are ADD and SUB. If the addnsub3 port is used, this parameter is ignored. If omitted, the default value is ADD.
ACCUMULATOR	String	No	Specifies the accumulator mode of the final adder stage. Values are YES and NO. If omitted, the default value is NO.
ACCUM_DIRECTION	String	No	Specifies whether the accumulator adds or subtracts its value from the previous sum. Values are ADD and SUB. If omitted, the default value is ADD.
OUTPUT_REGISTER	String	No	Specifies the clock signal for the output register. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGISTERED.
OUTPUT_ACLR	String	No	Specifies the asynchronous clear signal for the second adder register. Values are NONE, ACLR0, and ACLR1. If omitted, the default value is NONE.
PORT_SIGN[]	String	No	Parameter [A, B]. Specifies the corresponding sign[a,b] input port usage. Values are PORT_USED and PORT_UNUSED. If omitted, the default value is PORT_UNUSED.

Parameter Name	Type	Required	Description
ADDNSUB_MULTIPLIER_REGISTER[]	String	No	Parameter [1, 3]. Specifies the clock signal for the register on the corresponding addnsub[] input. Values are UNREGISTERED, CLOCK0, CLOCK1and CLOCK2. If the corresponding addnsub[] port is UNUSED, this parameter is ignored. If omitted, the default value is UNREGISTERED.
ADDNSUB_MULTIPLIER_ACLR[]	String	No	Parameter [1, 3]. Specifies the asynchronous clear signal for the first register on the corresponding addnsub[] input. Values are NONE, ACLR0 and ACLR1. If the corresponding addnsub[] port value is UNUSED, this parameter is ignored. If omitted, the default value is NONE.
PORT_ADDNSUB[]	String	No	Parameter [1, 3]. Specifies the usage of the corresponding addnsub[] input port. Values are PORT_USED and PORT_UNUSED. If omitted, the default value is PORT_UNUSED.
CHAINOUT_ADDER	String	No	Specifies the chainout mode of the final adder stage. Values are YES and NO. If omitted, the default value is NO.
WIDTH_CHAININ	Integer	No	Width of the chainin[] port. WIDTH_CHAININ equals WIDTH_RESULT if chainin port is used. If omitted, the default value is 1.
ACCUM_SLOAD_REGISTER	String	No	Specifies the clock source for the first register on the accum_sload or sload_accum input. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGISTERED .
ACCUM_SLOAD_ACLR	String	No	Specifies the asynchronous clear source for the first register on the accum_sload or sload_accum input. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE.
SCANOUTA_REGISTER	String	No	Specifies the clock source for the scanouta data bus registers. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGISTERED.
SCANOUTA_ACLR	String	No	Specifies the asynchronous clear source for the scanouta data bus registers. Values are NONE, ACLR0, ACLR1 and ACLR2. If omitted, the default value is NONE.
WIDTH_C	Integer	No	Width of the datac[] port.
WIDTH_COEF	Integer	No	Specifies the width of the constant value stored.

Parameter Name	Type	Required	Description
INPUT_REGISTER_C[0…3]	String	No	Specifies the clock port for the datac[] operand of the multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is UNREGISTERED. INPUT_REGISTER_C [1 … 3] must have similar values with INPUT_REGISTER_C [0].
INPUT_ACLR_C[0…3]	String	No	Specifies the asynchronous clear for the datac[] operand of the multiplier. Values are NONE, ACLR0, ACLR1. If omitted, the default value is NONE. The INPUT_ACLR_C [1 … 3] value must be set similar to the value of INPUT_ACLR_C0.
LOADCONST_VALUE	Integer	No	Preload constant value to complement accumulator mode. Values are 2^N where 0 < N < 64.
PREADDER_MODE	String	No	Specifies the mode of pre-adder settings to be used. Values are SIMPLE, COEF, INPUT, SQUARE, and CONSTANT. The default value is SIMPLE
PREADDER_DIRECTION_[]	String	No	Parameter [0…3]. Specifies whether the pre-adder of the corresponding multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.
COEFFSEL[]_REGISTER	String	No	Parameter [0…3]. Specifies the clock source for the coefficient inputs of the corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
COEFFSEL[]_ACLR	String	No	Specifies the asynchronous clear source for the coefficient inputs to the first multiplier. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE. The value must be set similar to the value of INPUT_ACLR_A0.
SYSTOLIC_DELAY1	String	No	Specifies the clock source for the systolic register inputs of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of OUTPUT_REGISTER or set as UNREGISTERED.

Parameter Name	Type	Required	Description
SYSTOLIC_DELAY3	String	No	Specifies the clock source for the systolic register inputs of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of OUTPUT_REGISTER or set as UNREGISTERED.
SYSTOLIC_ACLR1	String	No	Specifies the asynchronous clear source for the systolic register inputs of the first multiplier. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE. The value must be set similar to the value of OUTPUT_ACLR.
SYSTOLIC_ACLR3	String	No	Specifies the asynchronous clear source for the systolic register inputs of the third multiplier. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE. The value must be set similar to the value of OUTPUT_ACLR.
COEF0_[]	Integer	No	Specifies the coefficient value [0…7] for the inputs of the first multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
COEF1_[]	Integer	No	Specifies the coefficient value [0…7] for the inputs of the second multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
COEF2_[]	Integer	No	Specifies the coefficient value [0…7] for the inputs of the third multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
COEF3_[]	Integer	No	Specifies the coefficient value [0…7] for the inputs of the fourth multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
DOUBLE_ACCUMULATOR	String	No	Enables the double accumulator register. Values are YES and NO. This parameter is only available for family Arria V.
INPUT_A[0…3]_LATENCY_CLOCK	String	No	Specifies the clock signal for the pipeline register on the corresponding dataa[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is UNREGISTERED.
INPUT_A[0…3]_LATENCY_ACLR	String	No	Specifies the asynchronous clear signal for the pipeline register on the corresponding dataa[] port. Values are NONE, ACLR0, ACLR1. If omitted, the default value is NONE.

Parameter Name	Type	Required	Description
INPUT_B[0 … 3]_LATENCY_CLOCK	String	No	Specifies the clock signal for the pipeline register on the corresponding datab[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is UNREGISTERED.
INPUT_B[0…3]_LATENCY_ACLR	String	No	Specifies the asynchronous clear signal for the pipeline register on the corresponding datab[] port. Values are NONE, ACLR0, ACLR1. If omitted, the default value is NONE.
INPUT_C[0 … 3]_LATENCY_CLOCK	String	No	Specifies the clock signal for the pipeline register on the corresponding datac[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is UNREGISTERED.
INPUT_C[0…3]_LATENCY_ACLR	String	No	Specifies the asynchronous clear signal for the pipeline register on the corresponding datac[] port. Values are NONE, ACLR0, ACLR1. If omitted, the default value is NONE.
COEFFSEL[0…3]_LATENCY_CLOCK	String	No	Specifies the clock signal for the pipeline register on the corresponding coefficient inputs. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2.
COEFFSEL[0…3]_LATENCY_ACLR	String	No	Specifies the asynchronous clear signal for the pipeline register on the corresponding coefficient inputs. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE.
SIGNED_LATENCY_CLOCK_[]	String	No	Parameter [A, B]. Specifies the clock signal for the pipeline register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is UNREGISTERED.
SIGNED_LATENCY_ACLR_[]	String	No	Parameter [A, B]. Specifies the asynchronous clear signal for the pipeline register on the corresponding sign[] port. Values are NONE, ACLR0, and ACLR1. If omitted the default value is NONE.
ADDNSUB_MULTIPLIER_LATENCY_ CLOCK[]	String	No	Parameter [1, 3]. Specifies the clock signal for the pipeline register on the corresponding addnsub[] input. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGIS- TERED.

Parameter Name	Type	Required	Description
ADDNSUB_MULTIPLIER_LATENCY_ACLR[]	String	No	Parameter [1, 3]. Specifies the asynchronous clear signal for the pipeline register on the corresponding addnsub[] input. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE.
ACCUM_SLOAD_LATENCY_CLOCK	String	No	Specifies the clock signal for the pipeline register on the corresponding accum_sload or sload_accum input. Values are UNREGISTERED, CLOCK0, CLOCK1 and CLOCK2. If omitted, the default value is UNREGISTERED.
ACCUM_SLOAD_LATENCY_ACLR	String	No	Specifies the asynchronous clear signal for the pipeline register on the corresponding accum_sload or sload_accum input. Values are NONE, ACLR0 and ACLR1. If omitted, the default value is NONE.

Design Example: Implementing a Simple Finite Impulse Response (FIR) Filter

This design example uses the ALTMULT_ADD megafunction to implement a simple FIR filter as shown in the following equation. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

$$y(t) = \sum_{i=0}^{n-1} A(t-i)B(i)$$

n represents the number of taps, *A(t)* represents the sequence of input samples, and *B(i)* represents the filter coefficients.

The number of taps *(n)* can be any value, but this example is of a simple FIR filter with $n = 4$, which is called a 4-tap filter. To implement this filter, the coefficients of data *B* is loaded into the *B* registers in parallel and a shiftin register moves data *A(0)* to *A(1)* to *A(2)*, and so on. With a 4-tap filter, at a given time *(t)*, the sum of four products is computed. This function is implemented using the shift register chain option in the ALTMULT_ADD megafunction.

With reference to the equation, input *B* represents the coefficients and data *A* represents the data that is shifted into. The *A* input (data) is shifted in with the main clock, named clock0. The *B* input (coefficients) is loaded at the rising edge of clock1 with the enable signal held high.

The following design files can be found in **altmult_add_DesignExample.zip**:

fir_fourtap.qar (archived Quartus II design files)

altmult_add_ex_msim (ModelSim-Altera files)

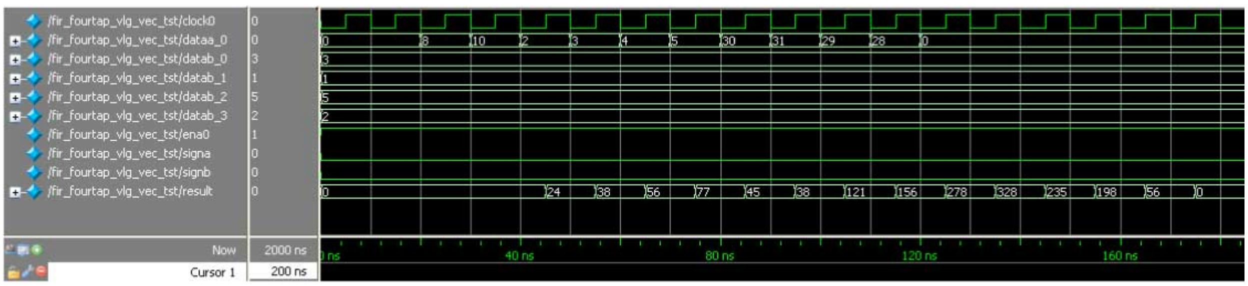
Understanding the Simulation Results

The following settings are observed in this example:

- The widths of the data inputs are all set to 16 bits
- The width of the output port, `result[]`, is set to 34 bits
- The input registers are all operating on the same clock

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 6-13: ALTMULT_ADD Simulation Results



ALTMEMMULT (Memory-based Constant Coefficient Multiplier)

7

2014.12.19

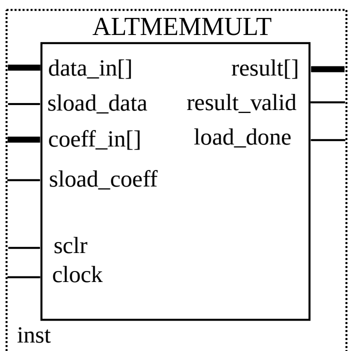
UG-01063 Subscribe Send Feedback

The ALTMEMMULT megafunction is used to create memory-based multipliers using the on-chip memory blocks found in Altera FPGAs (with M512, M4K, M9K, and MLAB memory blocks). This megafunction is useful if you do not have sufficient resources to implement the multipliers in logic elements (LEs) or dedicated multiplier resources.

The ALTMEMMULT megafunction is a synchronous function that requires a clock. The ALTMEMMULT megafunction and the MegaWizard Plug-In Manager create a multiplier with the smallest throughput and latency possible for a given set of parameters and specifications.

The following figure shows the ports for the ALTMEMMULT megafunction.

Figure 7-1: ALTMEMMULT Ports



Features

The ALTMEMMULT megafunction offers the following features:

- Creates only memory-based multipliers using on-chip memory blocks found in Altera FPGAs
- Supports data width of 1–512 bits
- Supports signed and unsigned data representation format
- Supports pipelining with fixed output latency
- Stores multiples constants in random-access memory (RAM)
- Provides an option to select the RAM block type
- Supports optional synchronous clear and load-control input ports

Resource Utilization and Performance

The following table provides resource utilization and performance information for the ALTMEMMULT megafunction.

Table 7-1: ALTMEMMULT Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			f_{MAX} (MHz) [4]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
Stratix III	data(4) × coeff(4)	2	61	62	41	445
	data(8) × coeff(8)	7	65	88	55	567
	data(16) × coeff(16)	7	151	175	107	445
Stratix IV	data(4) × coeff(4)	2	43	55	36	623
	data(8) × coeff(8)	7	65	88	56	605
	data(16) × coeff(16)	7	109	156	96	570

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (.v) **altera_mf.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module altmemmult
#( parameter coeff_representation = "SIGNED",
parameter coefficient0 = "UNUSED",
parameter data_representation = "SIGNED",
parameter intended_device_family = "unused",
parameter max_clock_cycles_per_result = 1,
parameter number_of_coefficients = 1,
parameter ram_block_type = "AUTO",
parameter total_latency = 1,
parameter width_c = 1,
parameter width_d = 1,
parameter width_r = 1,
parameter width_s = 1,
parameter lpm_type = "altmemmult",
parameter lpm_hint = "unused")
( input wire clock,
input wire [width_c-1:0]coeff_in,
input wire [width_d-1:0] data_in,
output wire load_done,
output wire [width_r-1:0] result,
output wire result_valid,
```

(4) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

```
input wire sclr,
input wire [width_s-1:0] sel,
input wire sload_coeff,
input wire sload_data)/* synthesis syn_black_box=1 */;
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altmemmult
generic (
coeff_representation:string := "SIGNED";
coefficient0:string := "UNUSED";
data_representation:string := "SIGNED";
intended_device_family:string := "unused";
max_clock_cycles_per_result:natural := 1;
number_of_coefficients:natural := 1;
ram_block_type:string := "AUTO";
total_latency:natural;
width_c:natural;
width_d:natural;
width_r:natural;
width_s:natural := 1;
lpm_hint:string := "UNUSED";
lpm_type:string := "altmemmult");
port(
clock:in std_logic;
coeff_in:in std_logic_vector(width_c-1 downto 0) := (others => '0');
data_in:in std_logic_vector(width_d-1 downto 0);
load_done:out std_logic;
result:out std_logic_vector(width_r-1 downto 0);
result_valid:out std_logic;
sclr:in std_logic := '0';
sel:in std_logic_vector(width_s-1 downto 0) := (others => '0');
sload_coeff:in std_logic := '0';
sload_data:in std_logic := '0');
end component;
```

Ports

The following tables list the input and output ports for the ALTMEMMULT megafunction.

Table 7-2: ALTMEMMULT Megafunction Input Ports

Port Name	Required	Description
`clock`	Yes	Clock input to the multiplier.
`coeff_in[]`	No	Coefficient input port for the multiplier. The size of the input port depends on the `WIDTH_C` parameter value.
`data_in[]`	Yes	Data input port to the multiplier. The size of the input port depends on the `WIDTH_D` parameter value.
`sclr`	No	Synchronous clear input. If unused, the default value is active high.

Port Name	Required	Description
sel[]	No	Fixed coefficient selection. The size of the input port depends on the WIDTH_S parameter value.
sload_coeff	No	Synchronous load coefficient input port. Replaces the current selected coefficient value with the value specified in the coeff_in input.
sload_data	No	Synchronous load data input port. Signal that specifies new multiplication operation and cancels any existing multiplication operation. If the MAX_CLOCK_CYCLES_PER_RESULT parameter has a value of 1, the sload_data input port is ignored.

Table 7-3: ALTMEMMULT Megafunction Output Ports

Port Name	Required	Description
result[]	Yes	Multiplier output port. The size of the input port depends on the WIDTH_R parameter value.
result_valid	Yes	Indicates when the output is the valid result of a complete multiplication. If the MAX_CLOCK_CYCLES_PER_RESULT parameter has a value of 1, the result_valid output port is not used.
load_done	No	Indicates when the new coefficient has finished loading. The load_done signal asserts when a new coefficient has finished loading. Unless the load_done signal is high, no other coefficient value can be loaded into the memory.

Parameters

The following table lists the parameters for the ALTMEMMULT megafunction.

Table 7-4: ALTMEMMULT Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH_D	Integer	Yes	Specifies the width of the data_in[] port.
WIDTH_C	Integer	Yes	Specifies the width of the coeff_in[] port.
WIDTH_R	Integer	Yes	Specifies the width of the result[] port.
WIDTH_S	Integer	No	Specifies the width of the sel[] port.
COEFFICIENT0	Integer	Yes	Specifies value of the first fixed coefficient.
TOTAL_LATENCY	Integer	Yes	Specifies the total number of clock cycles from the start of a multiplication to the time the result is available at the output.

Parameter Name	Type	Required	Description
DATA_REPRESENTATION	String	No	Specifies whether the coeff_in[] input port and the pre-loaded coefficients are signed or unsigned.
COEFF_REPRESENTATION	String	No	Specifies whether the coeff_in[] input port and the pre-loaded coefficients are signed or unsigned.
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the ALTMEMMULT megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES" The default value is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.
MAX_CLOCK_CYCLES_PER_RESULT	Integer	No	Specifies the number of clock cycles per result.
NUMBER_OF_COEFFICIENTS	Integer	No	Specifies the number of coefficients that are stored in the lookup table.
RAM_BLOCK_TYPE	String	No	Specifies the ram block type. Values are AUTO, SMALL, MEDIUM, M512, and M4K. If omitted, the default value is AUTO.

Design Example: 8 × 8 Multiplier

This design example uses the ALTMEMMULT megafunction to generate a basic multiplier using RAM blocks to determine the 16-bit product of two unsigned 8-bit numbers. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **altmemmult_DesignExample.zip**:

memmult_ex.qar (archived Quartus II design files)

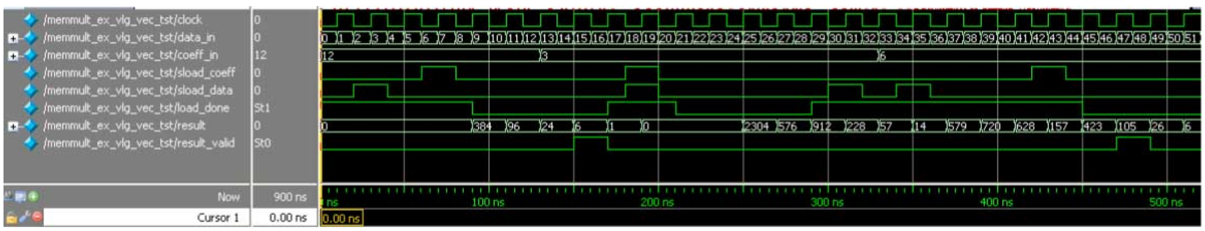
altmemmult_ex_msim (ModelSim-Altera files)

Understanding the Simulation Results

The following settings are observed in this example:

- The data_in[] and coeff_in[] input widths are both set to 8 bits
- The output port, result[] is set to a width of 16 bits
- The initial coefficient is 2
- The output latency is fixed to seven clock cycles based on the input widths set
- The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 7-2: ALTMEMMULT Simulation Results



This design example implements a multiplier for unsigned 8-bit numbers. If the value of the MAX_CLOCK_CYCLES_PER_RESULT parameter is more than 1, the sload_data signal indicates a new multiplication and the result_valid signal indicates the validity of the multiplication result.

If the value of the MAX_CLOCK_CYCLES_PER_RESULT parameter is 1, the sload_data signal is not used and every positive clock edge starts a new multiplication.

In this design example, with the MAX_CLOCK_CYCLES_PER_RESULT parameter set to 4, the design requires no less than four clock cycles to compute the multiplication. The sload_data signal is used to indicate a new multiplication.

Altera recommends that you do not pull the sload_data signal high during the four clock cycles when the multiplication is taking place to avoid getting unpredictable results.

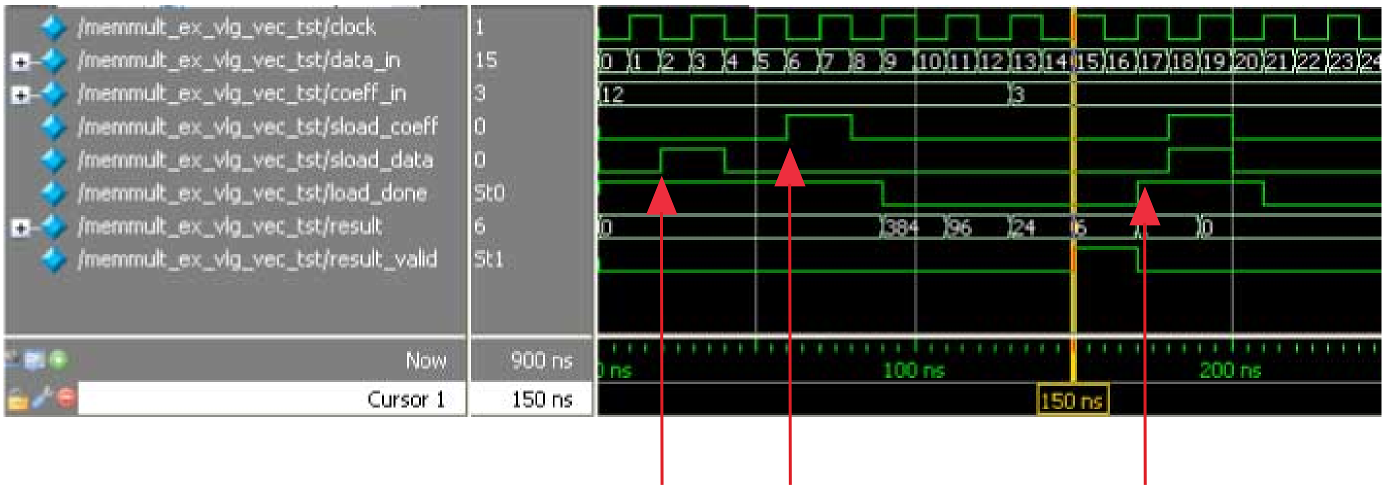
The megafunction only receive new inputs after four clock cycles. With the TOTAL_LATENCY parameter set to 7, all multiplication results require seven clock cycles to appear at the result[] port. The COEFFICIENT0 parameter holds the value of the first fixed coefficient, which is set to 2 (COEFFICIENT0=2) for this design example. The megafunction uses the latest coefficient value for every multiplication.

The sload_data signal asserts when a new coefficient value is written into the register. The load_done signal pulls low one clock cycle after the sload_data signal deasserts. When the load_done signal is low, the new coefficient value is reprogrammed into the RAM look-up table. Until the load_done signal pulls high, no other coefficient value can be loaded into the memory (regardless of whether the sload_data signal asserts anytime in between). The load_done signal asserts when programming completes.

The load time required to write a new coefficient value into the register is the same for any instance of the ALTMEMMULT megafunction. However, the load time can vary depending on the size of the RAM used.

The following figure shows the simulation results for the multiplication implementation with coefficient of 2.

Figure 7-3: Multiplication with Coefficient of 2



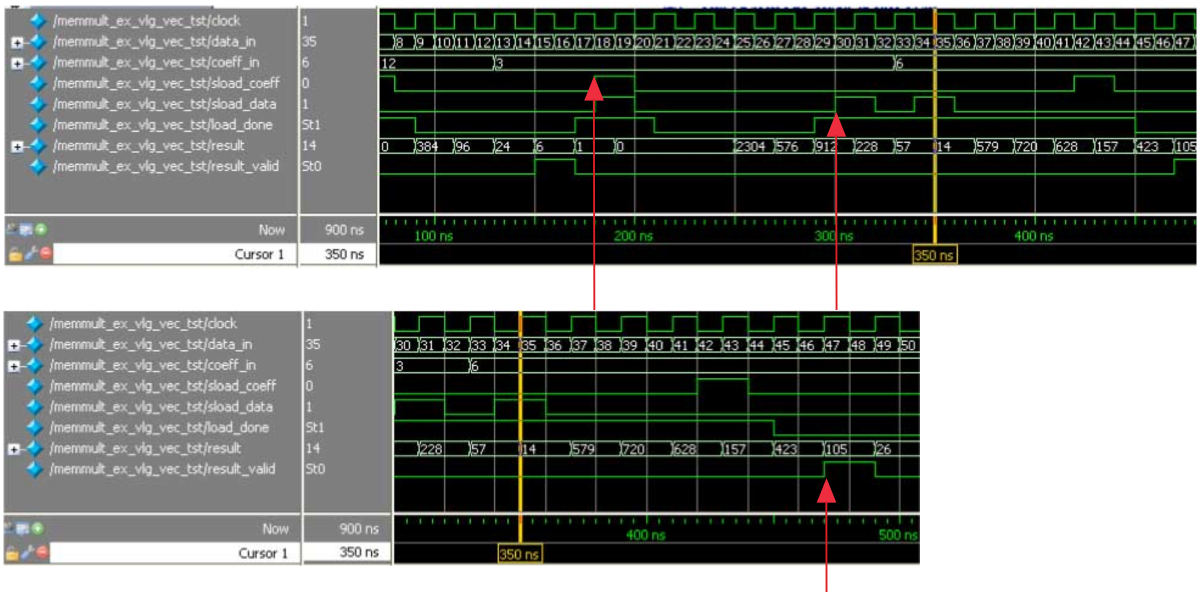
The following sequence corresponds with the numbered items in the figure:

1. The following sequence corresponds with the numbered items in the figure:

   At 30 ns, the sload_data signal asserts and triggers the first multiplication between the data_in value of 3 and the COEFFICIENT0 value of 2. The result is sent to the result port seven clock cycles later at 150 ns. The result_valid signal asserts to indicate that the multiplication is valid.
2. At 70 ns, the sload_coeff signal asserts to register a new coefficient value of 12 into the register.
3. The load_done signal pulls low to begin loading the new value into the memory, and pulls high at 170 ns when loading is complete. In this example, the load time is five clock cycles.

   The following figure shows the simulation results for the multiplication implementation with coefficient of 3.

Figure 7-4: Multiplication with Coefficient of 3



The following sequence corresponds with the numbered items in the figure:

1. At 190 ns, the sload_coeff signal asserts to register a new coefficient value of 3. The sload_data signal asserts and triggers a new multiplication. The latest value of the coefficient loaded into the memory is 12. Multiplication occurs between the data_in value of 19 and a coefficient of 12. At the same time, at 190 ns, the sload_coeff signal asserts and triggers the programming of coefficient 3. Although the multiplication result of 228 at 310 ns is valid, the result_valid signal does not pull high.
2. At 300 ns, the sload_data signal asserts and triggers a new multiplication. However, at 350 ns (less than four clock cycles after 300 ns), the sload_data signal pulls high again and cancels the previous multiplication process.
3. Multiplication finally occurs between the data_in value of 35 and a coefficient of 3 at 350 ns.The valid result, 105, of the computation is displayed at 470 ns.

Note: Altera recommends that you do not assert both the sload_coeff and sload_data signals at the same time to prevent the programming and computation processes from occurring simultaneously.

ALTMULT_ACCUM (Multiply-Accumulate) 8

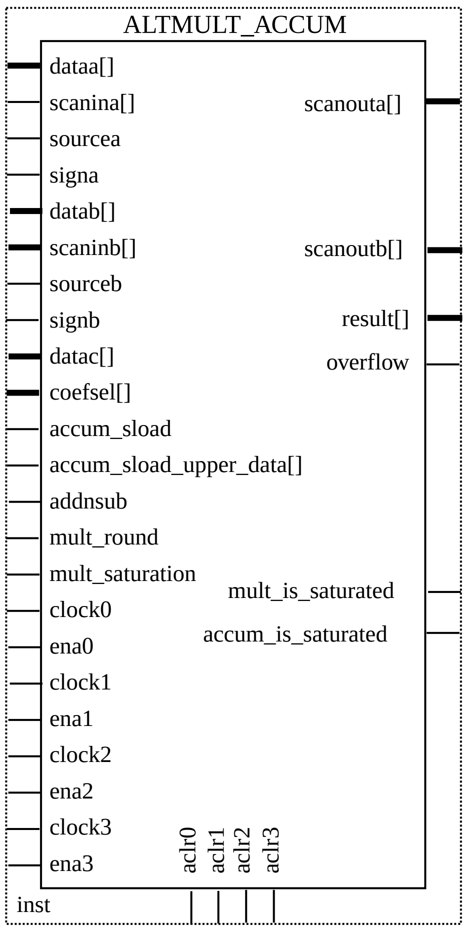
2014.12.19

UG-01063 Subscribe Send Feedback

The ALTMULT_ACCUM megafunction allows you to implement a multiplier-adder.

The following figure shows the ports for the ALTMULT_ACCUM megafunction.

Figure 8-1: ALTMULT_ACCUM Ports



A multiplier-accumulator accepts a pair of inputs, multiplies the two inputs together, and feeds their result into an accumulator to be added to or subtracted from its previous registered result. This function is expressed in the following equation.

$$y = \sum_{i-0}^{N-1} (\pm 1) \times A_i \times B_i$$

Where N is the number of cycles of data that has been entered into the accumulator.

Features

The ALTMULT_ACCUM megafunction offers the following features:

- Generates a multiplier-accumulator
- Supports data widths of 1–256 bits
- Supports signed and unsigned data representation format
- Supports pipelining with configurable output latency
- Provides a choice of implementation in dedicated DSP block circuitry or logic elements (LEs)

  Note: When building multipliers larger than the natively supported size there may/will be a performance impact resulting from the cascading of the DSP blocks.
- Provides an option to dynamically switch between add and subtract operations in the accumulator
- Provides an option to dynamically switch between signed and unsigned data support
- Provides an option to set up data shift register chains
- Supports hardware saturation and rounding (for Stratix III and Stratix IV devices only)
- Supports optional asynchronous clear and clock enable input ports
- Supports systolic delay register mode (for Arria V, Cyclone V, and Stratix V devices only)
- Supports pre-adder with 8 coefficients per multiplier (for Arria V, Cyclone V, and Stratix V devices only)
- Supports pre-load constant to complement accumulator feedback (for Arria V, Cyclone V, and Stratix V devices only)

Refer to the following megafunctions in this user guide for other multiplier implementations:

- Multiplier-Adder Megafunction (ALTMULT_ADD)
- Memory-based Constant Coefficient Multiplier (**ALTMEMMULT (Memory-based Constant Coefficient Multiplier)**)
- Multiplier Megafunction (**LPM_MULT (Multiplier)**)

Resource Utilization and Performance

The following table provides resource utilization and performance information for the ALTMULT_ACCUM megafunction.

Table 8-1: ALTMULT_ACCUM Resource Utilization and Performance

Device family	Input data width	Number of multipliers	Logic Usage			18-bit DSP	f_{MAX} (MHz) [5]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)		
Stratix III	16 ×16	1	0	0	0	4	481
	16 ×16	2	0	0	0	4	481
Stratix IV	16 ×16	1	0	0	0	4	443
	16 ×16	2	0	0	0	4	443

In the Stratix, Stratix GX, and Arria GX device series, the multiplier and the accumulator of the ALTMULT_ACCUM megafunction are placed in the DSP block circuitry. For Stratix V devices, the multiplier blocks and adder/accumulator block (mac_mult and mac_out) are combined into a single multiplier accumulator (MAC) block. The DSP blocks use the 18-bit × 18-bit input multiplier to process data with widths of up to 18 bits.

The registers and extra pipeline registers for the following signals are also placed inside the DSP block:

- Data input
- Signed or unsigned select
- Add or subtract select
- Synchronous load
- Products of multipliers

In the case of the output result, the first register is placed in the DSP block. The extra latency registers are placed in logic elements outside the block.

Cyclone II and Cyclone III devices have embedded multiplier blocks. When the ALTMULT_ACCUM megafunction is implemented in Cyclone II and Cyclone III devices, the multiplier is implemented in the embedded multiplier blocks, while the accumulator is put in LEs. In Cyclone devices, both the multiplier and accumulator are placed in LEs.

For more information about DSP blocks in any of the Stratix, Stratix GX, and Arria GX device series, refer to the DSP Blocks chapter of the respective handbooks on the **Literature and Technical Documentation page**.

For more information about the embedded memory blocks in any of the Stratix, Stratix GX, and Arria GX device series, refer to the TriMatrix Embedded Memory Blocks chapter of the respective handbooks on the **Literature and Technical Documentation page**.

For more information about embedded multiplier blocks in the Cyclone II and Cyclone III devices, refer to the DSP Blocks chapter of the respective handbooks on the **Literature and Technical Documentation page**.

For more information about implementing multipliers using DSP and memory blocks in Altera FPGAs, refer to **AN 306: Implementing Multipliers in FPGA Devices**.

[5] The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

Verilog HDL Prototype

To view the Verilog HDL prototype for the megafunction, refer to the Verilog Design File (**.v**) **altera_mf.v** in the **<*Quartus II installation directory*>\eda\synthesis** directory.

VHDL Component Declaration

To view the VHDL component declaration for the megafunction, refer to the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the **<*Quartus II installation directory*>\libraries\vhdl\altera_mf** directory.

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

ALTMULT_ACCUM Ports

The following tables list the input and output ports for the ALTMULT_ACCUM megafunction.

Note: For Arria V, Cyclone V, and Stratix V devices, each register can select between two asynchronous clear signals (`ACLR0`, `ACLR1`) and three clock/enable pairs (`CLOCK0/ENA0`, `CLOCK1/ENA1`, `CLOCK2/ENA2`).

Table 8-2: ALTMULT_ACCUM Megafunction Input Ports

Port Name	Required	Description
`accum_sload`	No	Causes the value on the accumulator feedback path to go to zero (0) or to `accum_sload_upper_data` when concatenated with 0. If the accumulator is adding and the `accum_sload` port is high, then the multiplier output is loaded into the accumulator. If the accumulator is subtracting, then the opposite (negative value) of the multiplier output is loaded into the accumulator. Beginning from Stratix V devices onwards, the `accum_sload` port causes the value on the accumulator feedback path to go to zero (0) or to `accum_sload_upper_data` when concatenated with 1 and loads the multiplier output if the accum_sload port is low.
`aclr0`	No	The first asynchronous clear input. The `aclr0` port is active high.
`aclr1`	No	The second asynchronous clear input. The `aclr1` port is active high.

Port Name	Required	Description
aclr2	No	The third asynchronous clear input. The aclr2 port is active high.
aclr3	No	The fourth asynchronous clear input. The aclr3 port is active high.
addnsub	No	Controls the functionality of the adder. If the addnsub port is high, the adder performs an add function; if the addnsub port is low, the adder performs a subtract function.
clock0	No	Specifies the first clock input, usable by any register in the megafunction.
clock1	No	Specifies the second clock input, usable by any register in the megafunction.
clock2	No	Specifies the third clock input, usable by any register in the megafunction.
clock3	No	Specifies the fourth clock input, usable by any register in the megafunction.
dataa[]	Yes	Data input to the multiplier. The size of the input port depends on the WIDTH_A parameter value.
datab[]	Yes	Data input to the multiplier. The size of the input port depends on the WIDTH_B parameter value.
ena0	No	Clock enable for the clock0 port.
ena1	No	Clock enable for the clock1 port.
ena2	No	Clock enable for the clock2 port.
ena3	No	Clock enable for the clock3 port.
signa	No	Specifies the numerical representation of the dataa[] port. If the signa port is high, the multiplier treats the dataa[] port as signed two's complement. If the signa port is low, the multiplier treats the dataa[] port as an unsigned number.
signb	No	Specifies the numerical representation of the datab[] port. If the signb port is high, the multiplier treats the datab[] port as signed two's complement. If the signb port is low, the multiplier treats the datab[]port as an unsigned number.

Table 8-3: ALTMULT_ACCUM Megafunction Input Ports (Stratix III and Stratix IV Devices Only)

Port Name	Required	Description
accum_round	No	Enables accumulator rounding.

Table 8-4: ALTMULT_ACCUM Megafunction Output Ports

Port Name	Required	Description
overflow	No	Overflow port for the accumulator.

Port Name	Required	Description
result[]	Yes	Accumulator output port. The size of the output port depends on the WIDTH_RESULT parameter value.
scanouta[]	No	Output of the first shift register. The size of the output port depends on the WIDTH_A parameter value. When instantiating the ALTMULT_ACCUM megafunction with the MegaWizard Plug-In Manager, the MegaWizard Plug-In Manager renames the scanouta[] port to shiftouta port.
scanoutb[]	No	Output of the second shift register. The size of the input port depends on the WIDTH_B parameter value. When instantiating the ALTMULT_ACCUM megafunction with the MegaWizard Plug-In Manager, the MegaWizard Plug-In Manager renames the scanoutb[] port to shiftoutb port.

ALTMULT_ACCUM Parameters

The following table lists the parameters for the ALTMULT_ACCUM megafunction.

Table 8-5: ALTMULT_ACCUM Megafunction Parameters

Parameter Name	Type	Required	Description
ACCUM_DIRECTION	String	No	Specifies whether the accumulator performs an add or subtract function. Values are ADD and SUB. When this parameter is set to ADD, the accumulator adds the product to the current accumulator value. When this parameter is set to SUB, the accumulator subtracts the product from the current accumulator value. If omitted the default value is ADD. This parameter is ignored if the addnsub port is used.
ACCUM_SLOAD_ACLR	String	No	Specifies the asynchronous clear signal for the accum_sload port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3. This parameter is ignored if the accum_sload port is unused.

Parameter Name	Type	Required	Description
ACCUM_SLOAD_PIPELINE_ACLR	String	No	Specifies the asynchronous clear signal for the second register on the accum_sload port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3. This parameter is ignored if the accum_sload port is unused.
ACCUM_SLOAD_PIPELINE_REG	String	No	Specifies the clock signal for the second register on the accum_sload port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the accum_sload port is unused.
ACCUM_SLOAD_REG	String	No	Specifies the clock signal for the accum_sload port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the accum_sload port is unused.
ADDNSUB_ACLR	String	No	Specifies the asynchronous clear for the addnsub port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR0. This parameter is ignored if the addnsub port is unused.
ADDNSUB_PIPELINE_ACLR	String	No	Specifies the asynchronous clear for the second register on the addnsub port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR0. This parameter is ignored if the addnsub port is unused.

Parameter Name	Type	Required	Description
ADDNSUB_PIPELINE_REG	String	No	Specifies the clock for the second register on the addnsub port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the addnsub port is unused.
ADDNSUB_REG	String	No	Specifies the clock for the addnsub port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the addnsub port is unused.
DSP_BLOCK_BALANCING	String	No	Specifies whether to use DSP block balancing. Values are UNUSED, Auto, DSP blocks, Logic Elements, Off, Simple 18-bit Multipliers, Simple Multipliers, and Width 18-bit Multipliers.
EXTRA_ACCUMULATOR_LATENCY	String	No	Adds the number of clock cycles of latency specified by the OUTPUT_REG parameter to the accumulator portion of the DSP block.
EXTRA_MULTIPLIER_LATENCY	Integer	No	Specifies the number of clock cycles of latency for the multiplier portion of the DSP block. If the MULTIPLIER_REG parameter is specified, then the specified clock port is used to add the latency. If the MULTIPLIER_REG parameter is set to UNREGISTERED, then the clock0 port is used to add the latency.
INPUT_ACLR_A	String	No	Specifies the asynchronous clear port for the dataa[] port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3.

Parameter Name	Type	Required	Description
INPUT_ACLR_B	String	No	Specifies the asynchronous clear port for the datab[] port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3.
INPUT_REG_A	String	No	Specifies the clock port for the dataa[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
INPUT_REG_B	String	No	Specifies the clock port for the datab[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is CLOCK0.
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the ALTMULT_ACCUM megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES" The default value is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.

Parameter Name	Type	Required	Description
MULTIPLIER_ACLR	String	No	Specifies the asynchronous clear signal for the register immediately following the multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3.
MULTIPLIER_REG	String	No	Specifies the clock signal for the register that immediately follows the multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
OUTPUT_ACLR	String	No	Specifies the asynchronous clear signal for the registers on the outputs. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3.
OUTPUT_REG	String	No	Specifies the clock signal for the registers on the outputs. Values are CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted the default value is CLOCK0.
PORT_ADDNSUB	String	No	Specifies the usage of the addnsub input port. Values are: PORT_USED , PORT_UNUSED, and PORT_CONNECTIVITY (port usage is determined by checking the port connectivity.) If omitted the default value is PORT_CONNECTIVITY.
PORT_SIGNA	String	No	Specifies the usage of the signa input port. Values are PORT_USED, PORT_UNUSED, and PORT_CONNECTIVITY. If omitted the default value is PORT_CONNECTIVITY. Beginning from Stratix V devices onwards, the PORT_CONNECTIVITY parameter is not supported. The default value is PORT_UNUSED.

Parameter Name	Type	Required	Description
PORT_SIGNB	String	No	Specifies the usage of the signb input port. Values are PORT_USED, PORT_UNUSED, and PORT_CONNECTIVITY. If omitted the default value is PORT_CONNECTIVITY. Beginning from Stratix V devices onwards, the PORT_CONNECTIVITY parameter is not supported. The default value is PORT_UNUSED.
REPRESENTATION_[]	String	No	Parameter [A,B]. Specifies the numerical representation of the corresponding data[] port. Values are UNSIGNED and SIGNED. When this parameter is set to SIGNED, the accumulator interprets the dataa input as signed two's complement. If omitted, the default value is UNSIGNED. This parameter is ignored if the signa port is used.
SIGN_ACLR_[]	String	No	Parameter [A,B]. Specifies the asynchronous clear signal for the first register on the corresponding sign[] port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3. This parameter is ignored if the corresponding sign[] port is unused.
SIGN_PIPELINE_ACLR_[]	String	No	Parameter [A,B]. Specifies the asynchronous clear signal for the second register on the corresponding sign[] port. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted the default value is ACLR3. This parameter is ignored if the corresponding sign[] port is unused.

Parameter Name	Type	Required	Description
SIGN_PIPELINE_REG_[]	String	No	Parameter [A,B]. Specifies the clock signal for the second register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the corresponding sign[] port is unused.
SIGN_REG_[]	String	No	Parameter [A,B]. Specifies the clock signal for the first register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. This parameter is ignored if the corresponding sign[] port is unused.
WIDTH_A	Integer	Yes	Specifies the width of the dataa[] port.
WIDTH_B	Integer	Yes	Specifies the width of the datab[] port.
WIDTH_RESULT	Integer	No	Specifies the width of the result[] port.

Table 8-6: ALTMULT_ACCUM Megafunction Parameters (Stratix II, Stratix II GX, Stratix III, Stratix IV, Arria GX, and HardCopy devices only)

Parameter Name	Type	Required	Description
MULTIPLIER_SATURATION	String	No	Specifies multiplier saturation. Values are NO, YES, and VARIABLE. If omitted the default value is NO.

Table 8-7: ALTMULT_ACCUM Megafunction Parameters (Arria V, Cyclone V, and Stratix V Devices Only)

Parameter Name	Type	Required	Description
INPUT_ACLR_C0	String	No	Specifies the asynchronous clear for the datac[] operand of the first multiplier. Values are ACLR0 and ACLR1. If omitted and corresponding INPUT_REGISTER_C[] is used, the default value is ACLR0. The value for INPUT_ACLR_C0 must be set similar to the value of INPUT_ACLR_A0.
INPUT_ACLR_C1	String	No	Specifies the asynchronous clear for the datac[] operand of the second multiplier. Values are ACLR0 and ACLR1. If omitted and corresponding INPUT_REGISTER_C[] is used, the default value is ACLR0. The value for INPUT_ACLR_C1 must be set similar to the value of INPUT_ACLR_A0.
INPUT_ACLR_C2	String	No	Specifies the asynchronous clear for the datac[] operand of the third multiplier. Values are ACLR0 and ACLR1. If omitted and corresponding INPUT_REGISTER_C[] is used, the default value is ACLR0. The value for INPUT_ACLR_C2 must be set similar to the value of INPUT_ACLR_A0.
INPUT_ACLR_C3	String	No	Specifies the asynchronous clear for the datac[] operand of the fourth and corresponding multiplier. Values are ACLR0 and ACLR1. If omitted and corresponding INPUT_REGISTER_C[] is used, the default value is ACLR0. The value for INPUT_ACLR_C3 must be set similar to the value of INPUT_ACLR_A0.

Parameter Name	Type	Required	Description
INPUT_REGISTER_C0	String	No	Specifies the clock port for the datac[] operand of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is CLOCK0. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
INPUT_REGISTER_C1	String	No	Specifies the clock port for the datac[] operand of the second multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is CLOCK0. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
INPUT_REGISTER_C2	String	No	Specifies the clock port for the datac[] operand of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is CLOCK0. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
INPUT_REGISTER_C3	String	No	Specifies the clock port for the datac[] operand of the fourth and corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. If omitted, the default value is CLOCK0. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
MUTIPLIER1_DIRECTION	String	No	Specifies whether the second multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If the addnsub1 port is used, this parameter is ignored. If omitted, the default value is ADD.

Parameter Name	Type	Required	Description
MUTIPLIER3_DIRECTION	String	No	Specifies whether the fourth and all subsequent odd-numbered multipliers add or subtract their results from the total. Values are ADD and SUB. If the addnsub3 port is used, this parameter is ignored. If omitted, the default value is ADD.
NUMBER_OF_MULTIPLIERS	Integer	Yes	Number of multipliers to be added together. Values are 1 up to 4.
WIDTH_C	Integer	Yes	Specifies the width of the datac[] port.
WIDTH_COEF	Integer	Yes	Specifies the width of the coefsel[] port.
LOADCONST_VALUE	Integer	No	Pre-load constant value to complement accumulator mode. Values are 2^N where $0 < N < 64$.
PREADDER_MODE	String	No	Specifies the mode of pre-adder settings to be used. Values are SIMPLE, COEF, INPUT, SQUARE, and CONSTANT. The default value is SIMPLE.
PREADDER_DIRECTION_0	String	No	Specifies whether the pre-adder of the first multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.
PREADDER_DIRECTION_1	String	No	Specifies whether the pre-adder of the second multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.
PREADDER_DIRECTION_2	String	No	Specifies whether the pre-adder of the third multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.

Parameter Name	Type	Required	Description
PREADDER_DIRECTION_3	String	No	Specifies whether the pre-adder of the fourth and corresponding multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.
COEFFSEL0_REGISTER	String	No	Specifies the clock source for the coefficient inputs of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
COEFFSEL1_REGISTER	String	No	Specifies the clock source for the coefficient inputs of the second multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
COEFFSEL2_REGISTER	String	No	Specifies the clock source for the coefficient inputs of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.
COEFFSEL3_REGISTER	String	No	Specifies the clock source for the coefficient inputs of the fourth and corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of INPUT_REGISTER_A0 or set as UNREGISTERED.

Parameter Name	Type	Required	Description
COEFFSEL_A_ACLR	String	No	Specifies the asynchronous clear source for the coefficient inputs to the first multiplier. Values are NONE, ACLR0 and ACLR1. If omitted and corresponding COEFFSEL[]_REGISTER is used, the default value is ACLR0. The value must be set similar to the value of INPUT_ ACLR_A0.
COEFFSEL_B_ACLR	String	No	Specifies the asynchronous clear source for the coefficient inputs to the second multiplier. Values are NONE, ACLR0 and ACLR1. If omitted and corresponding COEFFSEL[]_REGISTER is used, the default value is ACLR0. The value must be set similar to the value of INPUT_ ACLR_A0.
COEFFSEL_C_ACLR	String	No	Specifies the asynchronous clear source for the coefficient inputs to the third multiplier. Values are NONE, ACLR0 and ACLR1. If omitted and the corresponding COEFFSEL[]_REGISTER is used, the default value is ACLR0. The value must be set similar to the value of INPUT_ ACLR_A0.
COEFFSEL_D_ACLR	String	No	Specifies the asynchronous clear source for the coefficient inputs to the fourth and corresponding multiplier. Values are NONE, ACLR0 and ACLR1. If omitted and the corresponding COEFFSEL[]_REGISTER is used, the default value is ACLR0. The value must be set similar to the value of INPUT_ ACLR_A0.

Parameter Name	Type	Required	Description
SYSTOLIC_DELAY1	String	No	Specifies the clock source for the systolic register inputs of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of OUTPUT_REGISTER or set as UNREGISTERED.
SYSTOLIC_DELAY3	String	No	Specifies the clock source for the systolic register inputs of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, and CLOCK2. The value must be set similar to the value of OUTPUT_REGISTER or set as UNREGISTERED.
SYSTOLIC_ACLR1	String	No	Specifies the asynchronous clear source for the systolic register inputs of the first multiplier. Values are NONE, ACLR0, and ACLR1. If omitted and the corresponding SYSTOLIC_DELAY[] is used, the default value is ACLR0. The value must be set similar to the value of OUTPUT_ACLR.
SYSTOLIC_ACLR3	String	No	Specifies the asynchronous clear source for the systolic register inputs of the third multiplier. Values are NONE, ACLR0, and ACLR1. If omitted and the corresponding SYSTOLIC_DELAY[] is used, the default value is ACLR0. The value must be set similar to the value of OUTPUT_ACLR.
COEF0_[]	Integer	No	Specifies the coefficient value [0..7] for the inputs of the first multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.

Parameter Name	Type	Required	Description
COEF1_[]	Integer	No	Specifies the coefficient value [0..7] for the inputs of the second multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
COEF2_[]	Integer	No	Specifies the coefficient value [0..7] for the inputs of the third multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.
COEF3_[]	Integer	No	Specifies the coefficient value [0..7] for the inputs of the fourth multiplier. The number of coefficient bits must be set similar to the value of WIDTH_COEF.

Design Example: Shift Accumulator

A multiplier-accumulator can be used to implement FIR filters. This design example uses the ALTMULT_ACCUM megafunction to implement a serial FIR filter, in which both the data and coefficient are shifted serially into the multiplier and then summed in the accumulator. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **altmult_accum_DesignExample.zip**:

serial_fir.qar (archived Quartus II design files)

altmult_accum_ex_msim (ModelSim-Altera files)

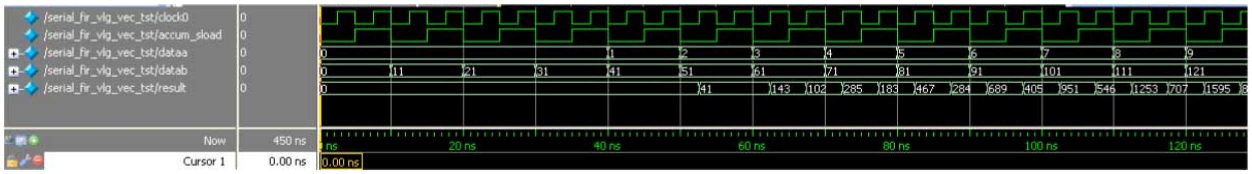
Understanding the Simulation Results

The following settings are observed in this example:

- The dataa[] and datab[] input widths are both set to 16 bits
- The output port, result[] is set to a width of 33 bits
- The accum_sload input is enabled

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 8-2: ALTMULT_ACCUM Simulation Results



ALTMULT_ADD (Multiply-Adder) 9

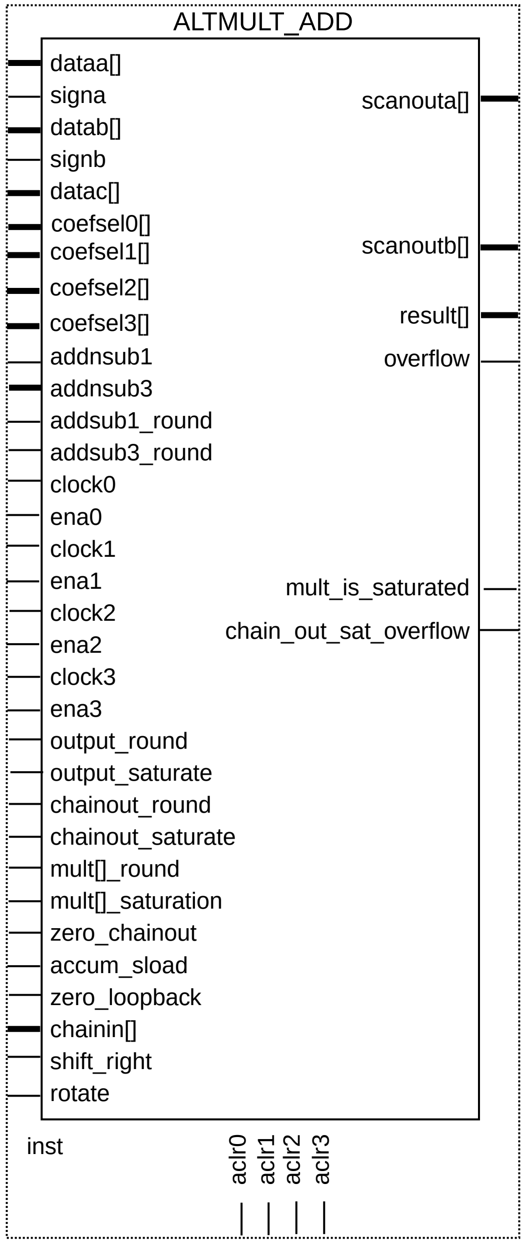
2014.12.19

UG-01063 Subscribe Send Feedback

The ALTMULT_ADD megafunction allows you to implement a multiplier-adder.

The following figure shows the ports for the ALTMULT_ADD megafunction.

Figure 9-1: ALTMULT_ADD Ports



A multiplier-adder accepts pairs of inputs, multiplies the values together and then adds to or subtracts from the products of all other pairs.

The ALTMULT_ADD megafunction also offers many variations in dedicated DSP block circuitry. Data input sizes of up to 18 bits are accepted. Because the DSP blocks allow for one or two levels of 2-input add or subtract operations on the product, this function creates up to four multipliers.

Stratix III and Stratix IV device families use two MAC blocks (mac_mult and mac_out) to form DSP operations, multiply and add. For Stratix V devices, the multiplier blocks and adder/accumulator block is combined in a single MAC block.

The multipliers and adders of the ALTMULT_ADD megafunction are placed in the dedicated DSP block circuitry of the Stratix devices. If all of the input data widths are 9-bits wide or smaller, the function uses the 9 × 9-bit input multiplier configuration in the DSP block. If not, the DSP block uses 18 × 18-bit input multipliers to process data with widths between 10 bits and 18 bits. If multiple ALTMULT_ADD megafunctions occur in a design, the functions are distributed to as many different DSP blocks as possible so that routing to these blocks is more flexible. Fewer multipliers per DSP block allow more routing choices into the block by minimizing paths to the rest of the device.

The registers and extra pipeline registers for the following signals are also placed inside the DSP block:

- Data input
- Signed or unsigned select
- Add or subtract select
- Products of multipliers

In the case of the output result, the first register is placed in the DSP block. However the extra latency registers are placed in logic elements outside the block. Peripheral to the DSP block, including data inputs to the multiplier, control signal inputs, and outputs of the adder, use regular routing to communicate with the rest of the device. All connections in the function use dedicated routing inside the DSP block. This dedicated routing includes the shift register chains when you select the option to shift a multiplier's registered input data from one multiplier to an adjacent multiplier.

For more information about implementing multipliers using DSP and memory blocks in Altera FPGAs, refer to **AN 306: Implementing Multipliers in FPGA Devices**.

Features

The ALTMULT_ADD megafunction offers the following features:

- Generates a multiplier to perform multiplication operations of two complex numbers
- Supports data widths of 1– 256 bits
- Supports signed and unsigned data representation format
- Supports pipelining with configurable output latency
- Provides a choice of implementation in dedicated DSP block circuitry or logic elements (LEs)

  Note: When building multipliers larger than the natively supported size there may/will be a performance impact resulting from the cascading of the DSP blocks.
- Provides an option to dynamically switch between signed and unsigned data support
- Provides an option to dynamically switch between add and subtract operation
- Provides an option to set up data shifting register chains
- Supports hardware saturation and rounding (for selected device families only)
- Supports optional asynchronous clear and clock enable input ports
- Supports systolic delay register mode (for Arria V, Cyclone V, and Stratix V devices only)
- Supports pre-adder with 8 pre-load coefficients per multiplier (for Arria V, Cyclone V, and Stratix V devices only)
- Supports pre-load constant to complement accumulator feedback (for Arria V, Cyclone V, and Stratix V devices only)
- In Arria V, Cyclone V, and Stratix V devices, the pre-adder, coefficient storage and systolic delay register features are added to maximize flexibility. The following sections describe the new features.

Pre-adder

With pre-adder, additions or subtractions are done prior to feeding the multiplier.

There are five pre-adder modes:

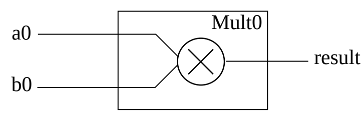
- Simple mode
- Coefficient mode
- Input mode
- Square mode
- Constant mode

Note: When pre-adder is used (pre-adder coefficient/input/square mode), all data inputs to the multiplier must have the same clock setting.

Pre-adder Simple Mode

In this mode, both operands derive from the input ports and pre-adder is not used or bypassed. This is the default mode.

Figure 9-2: Pre-adder Simple Mode



Pre-adder Coefficient Mode

In this mode, one multiplier operand derives from the pre-adder, and the other operand derives from the internal coefficient storage. The coefficient storage allows up to 8 preset constants. The coefficient selection signals are `coefsel[0..3]`.

The following settings are applied in this mode:

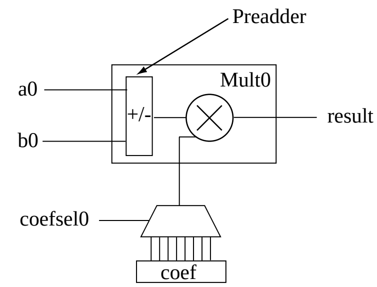
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 25 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 25 bits
- The width of the coefficient input must be less than or equals to 27 bits

This mode is expressed in the following equation.

$$y = (a + b) \times coef$$

The following shows the pre-adder coefficient mode of a multiplier.

Figure 9-3: Pre-adder Coefficient Mode



Pre-adder Input Mode

In this mode, one multiplier operand derives from the pre-adder, and the other operand derives from the `datac[]` input port.

The following settings are applied in this mode:

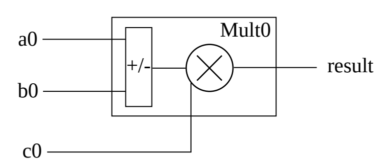
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 25 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 25 bits
- The width of the `datac[]` input (`WIDTH_C`) must be less than or equals to 22 bits
- The number of multipliers must be set to 1
- All input registers must be registered with the same clock

This mode is expressed in the following equation.

$$y = (a + b) \times c$$

The following shows the pre-adder input mode of a multiplier.

Figure 9-4: Pre-adder Input Mode



Pre-adder Square Mode

In this mode, both multiplier operands derive from the pre-adder.

The following settings are applied in this mode:

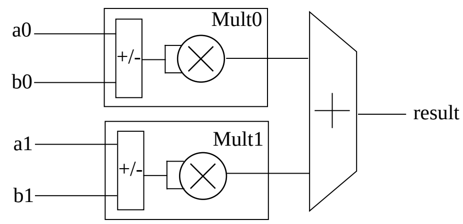
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 17 bits
- The width of the `datab[]` input (`WIDTH_B`) must be less than or equals to 17 bits
- The number of multipliers must be set to 2

This mode is expressed in the following equation.

$$y = (a0 + b0)^2 + (a1 + b1)^2$$

The following shows the pre-adder square mode of two multipliers.

Figure 9-5: Pre-adder Square Mode



Pre-adder Constant Mode

In this mode, one multiplier operand derives from the input port, and the other operand derives from the internal coefficient storage. The coefficient storage allows up to 8 preset constants. The coefficient selection signals are `coefsel[0..3]`.

The following settings are applied in this mode:

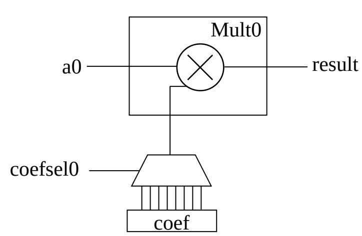
- The width of the `dataa[]` input (`WIDTH_A`) must be less than or equals to 27 bits
- The width of the coefficient input must be less than or equals to 27 bits
- The `datab[]` port must be disconnected

This mode is expressed in the following equation.

$$y = a0 \times coef$$

The following figure shows the pre-adder constant mode of a multiplier.

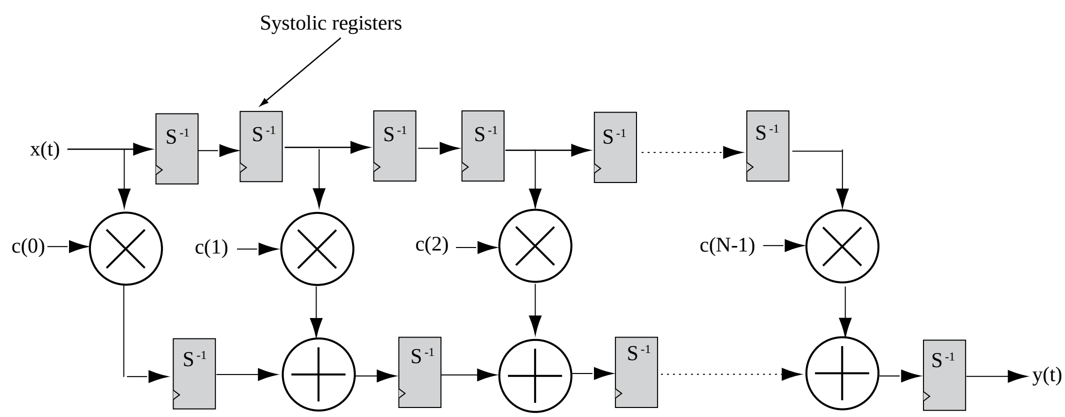
Figure 9-6: Pre-adder Constant Mode



Systolic Delay Register

In a systolic architecture, the input data is fed into a cascade of registers acting as a data buffer. Each register delivers an input sample to a multiplier where it is multiplied by the respective coefficient. The chain adder stores the gradually combined results from the multiplier and the previously registered result from the `chainin[]` input port to form the final result. Each multiply-add element must be delayed by a single cycle so that the results synchronize appropriately when added together. Each successive delay is used to address both the coefficient memory and the data buffer of their respective multiply-add elements. For example, a single delay for the second multiply add element, two delays for the third multiply-add element, and so on.

Figure 9-7: Systolic Registers



x(t) represents the results from a continuous stream of input samples and *y(t)* represents the summation of a set of input samples, and in time, multiplied by their respective coefficients. Both the input and output results flow from left to right. The *c(0)* to *c(N*-1) denotes the coefficients. The systolic delay registers are denoted by S^{-1}, whereas the $^{-1}$ represents a single clock delay. Systolic delay registers are added at the inputs and outputs for pipelining in a way that ensures the results from the multiplier operand and the accumulated sums stay in synch. This processing element is replicated to form a circuit that computes the filtering function. This function is expressed in the following equation.

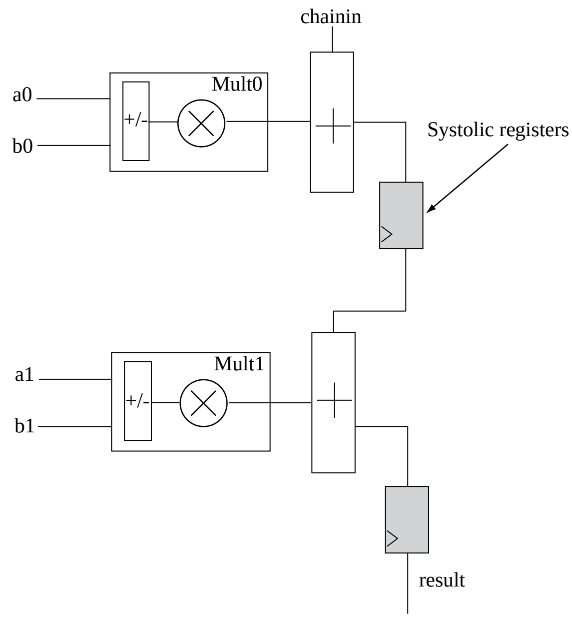
$$y(t) = \sum_{i=0}^{N-1} B(i)A(t-i)$$

N represents the number of cycles of data that has entered into the accumulator, *y(t)* represents the output at time *t*, *A(t)* represents the input at time *t*, and *B(i)* are the coefficients. The *t* and *i* in the equation correspond to a particular instant in time, so to compute the output sample *y(t)* at time t, a group of input samples at *N* different points in time, or A(n), A(n-1), A(n-2), … A(n-N+1) is required. The group of N input samples are multiplied by *N* coefficients and summed together to form the final result *y*.

The systolic register architecture is available only for sum-of-2 and sum-of-4 modes.

The following figure shows the systolic delay register implementation of 2 multipliers.

Figure 9-8: Systolic Delay Register Implementation of 2 Multipliers

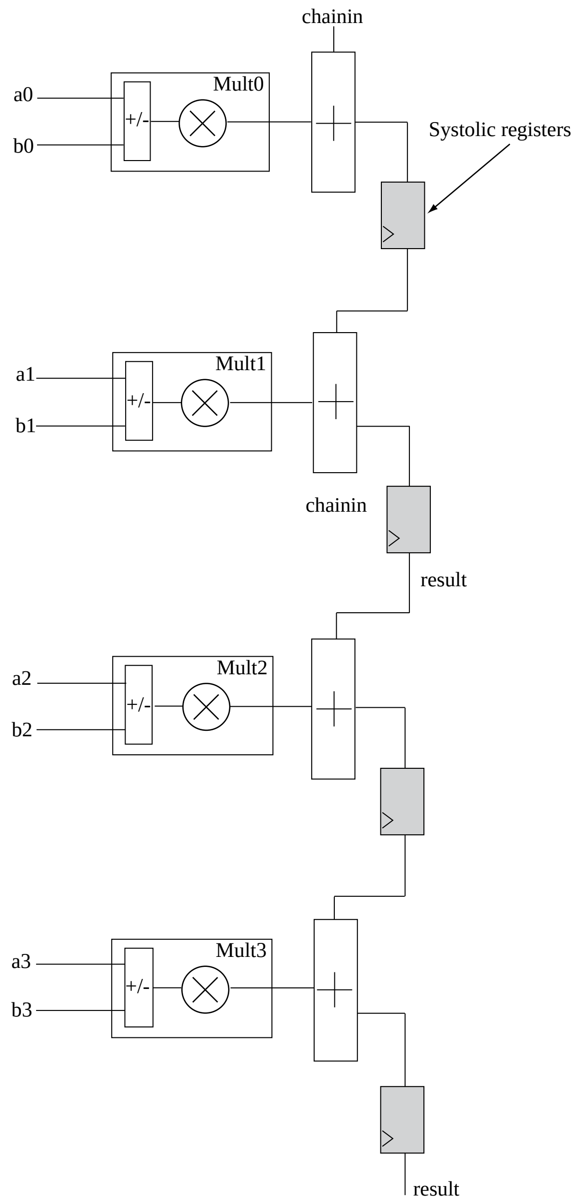


The sum of two multipliers is expressed in the following equation.

$$y(t) = [a1(t) \times b1(t)] + [a0(t-1) \times b0(t-1)]$$

The following figure shows the systolic delay register implementation of 4 multipliers.

Figure 9-9: Systolic Delay Register Implementation of 4 Multipliers



The sum of four multipliers is expressed in the following equation.

Figure 9-10: Sum of 4 Multipliers

$$y(t) = [a3(t) \times b3(t)] + [a2(t-1) \times b2(t-1)] + [a1(t-2) \times b1(t-2)] + [a0(t-3) \times b0(t-3)]$$

The following lists the advantages of systolic register implementation:

- Reduces DSP resource usage
- Enables efficient mapping in the DSP block using the chain adder structure

The systolic delay implementation is only available for the following pre-adder modes:

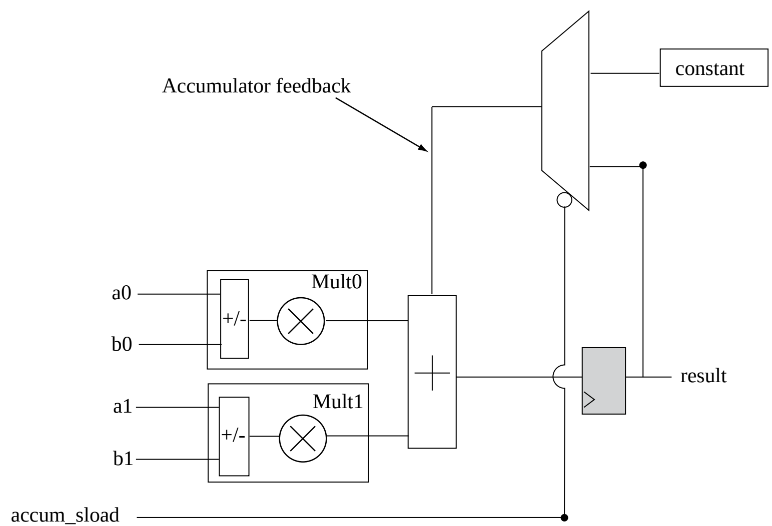
- Pre-adder coefficient mode
- Pre-adder simple mode
- Pre-adder constant mode

Pre-load Constant

The pre-load constant controls the accumulator operand and complements the accumulator feedback. The valid `LOADCONST_VALUE` ranges from 0–64. The constant value is equal to 2^N, where N = `LOADCONST_VALUE`. When the `LOADCONST_VALUE` is set to 64, the constant value is equal to 0. This function can be used as biased rounding.

The following figure shows the pre-load constant implementation.

Figure 9-11: Pre-load Constant



Refer to the following megafunctions in this user guide for other multiplier implementations:

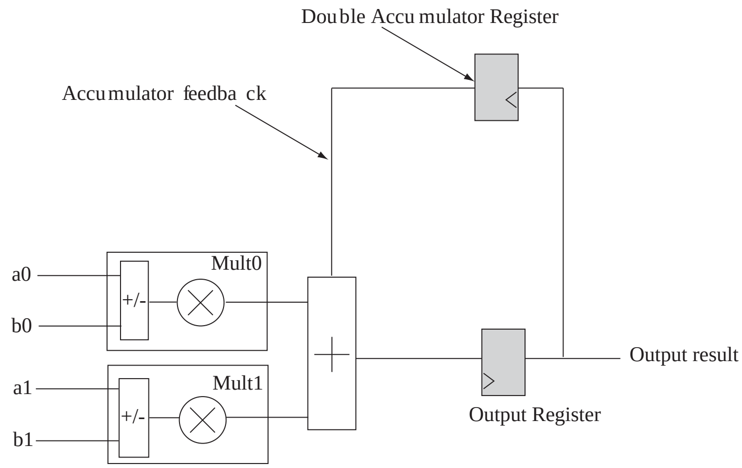
- **ALTMULT_ACCUM (Multiply-Accumulate)**
- **ALTMEMMULT (Memory-based Constant Coefficient Multiplier)**
- **LPM_MULT (Multiplier)**

Double Accumulator

The double accumulator feature adds an additional register in the accumulator feedback path. The double accumulator register follows the output register, which includes the clock, clock enable, and aclr. The additional accumulator register returns result with a one-cycle delay. This feature enables you to have two accumulator channels with the same resource count.

The following figure shows the double accumulator implementation.

Figure 9-12: Double Accumulator



Resource Utilization and Performance

The following table provides resource utilization and performance information for the ALTMULT_ADD megafunction.

Table 9-1: ALTMULT_ADD Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			18-bit DSP	f_{MAX} (MHz) [(6)]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)		
Stratix III	16 x 16	3	0	0	0	2	645
	32 x 32	3	0	0	0	4	454
	64 x 64	3	217	128	146	16	145

Verilog HDL Prototype

To view the Verilog HDL prototype for the megafunction, refer to the Verilog Design File (**.v**) **altera_mf.v** in the **<*Quartus II installation directory*>\eda\synthesis** directory.

VHDL Component Declaration

(6) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

To view the VHDL component declaration for the megafunction, refer to the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the ***<Quartus II installation directory>*\libraries\vhdl\altera_mf** directory.

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

ALTMULT_ADD Ports

The following tables list the input and output ports for the ALTMULT_ADD megafunction.

Note: For Arria V, Cyclone V, and Stratix V devices, each register can only select between two asynchronous signals (`ACLR0`, `ACLR1`) and three clock/enable pairs (`CLOCK0/ENA0`, `CLOCK1/ENA1`, `CLOCK2/ENA2`).

Table 9-2: ALTMULT_ADD Megafunction Input Ports

Port Name	Required	Description
`dataa[]`	Yes	Data input to the multiplier. Input port `[NUMBER_OF_MULTIPLIERS * WIDTH_A - 1..0]` wide.
`datab[]`	Yes	Data input to the multiplier. Input port `[NUMBER_OF_MULTIPLIERS * WIDTH_B - 1..0]` wide.
`clock[]`	No	Clock input port `[0..3]` to the corresponding register. This port can be used by any register in the megafunction.
`aclr[]`	No	Input port `[0..3]`. Asynchronous clear input to the corresponding register.
`ena[]`	No	Input port `[0..3]`. Clock enable for the corresponding clock[] port.
`signa`	No	Specifies the numerical representation of the `dataa[]` port. If the `signa` port is high, the multiplier treats the `dataa[]` port as a signed two's complement number. If the `signa` port is low, the multiplier treats the `dataa[]` port as an unsigned number.
`signb`	No	Specifies the numerical representation of the `datab[]` port. If the `signb` port is high, the multiplier treats the `datab[]` port as a signed two's complement number. If the `signb` port is low, the multiplier treats the `datab[]` port as an unsigned number.

Table 9-3: ALTMULT_ADD Megafunction Input Ports (Stratix III and Stratix IV Devices Only)

Port Name	Required	Description
Ports Available in Stratix III and Stratix IV devices only		

Port Name	Required	Description
output_round	No	Enables dynamically controlled output rounding. When OUTPUT_ROUNDING is set to VARIABLE, output_round enables the final adder stage of rounding.
output_saturate	No	Enables dynamically controlled output saturation. When OUTPUT_SATURATION is set to VARIABLE, output_saturate enables the final adder stage of saturation.
chainout_round	No	Enables dynamically controlled chainout stage rounding. When CHAINOUT_ROUNDING is set to VARIABLE, chainout_round enables the chainout stage of rounding.
chainout_saturate	No	Enables dynamically controlled chainout stage saturation. When CHAINOUT_SATURATION is set to VARIABLE, chainout_saturate enables the chainout stage of saturation.
zero_chainout	No	Dynamically specifies whether the chainout value is zero.
zero_loopback	No	Dynamically specifies whether the loopback value is zero.
accum_sload	No	Dynamically specifies whether the accumulator value is zero.
chainin	No	Adder result input bus from the preceding stage. Input port [WIDTH_CHAININ - 1..0] wide.
rotate	No	Specifies dynamically controlled port rotation in shift mode.
shift_right	No	Specifies dynamically controlled port shift right or left in shift mode. Values are 0 and 1. A value of 0 specifies a shift to the left, a value of 1 specifies a shift to the right.

Table 9-4: ALTMULT_ADD Megafunction Input Ports (Arria V, Cyclone V, and Stratix V Devices Only)

Port Name	Required	Description
datac[]	Yes	Data input to the multiplier. Input port [NUMBER_OF_MULTIPLIERS * WIDTH_C - 1..0] wide.
coefsel0[]	No	Coefficient input port[0..3] to the first multiplier.
coefsel1[]	No	Coefficient input port[0..3]to the second multiplier.
coefsel2[]	No	Coefficient input port[0..3]to the third multiplier.
coefsel3[]	No	Coefficient input port[0..3]to the fourth multiplier.

Table 9-5: ALTMULT_ADD Megafunction Output Ports

Port Name	Required	Description
result[]	Yes	Multiplier output port. Output port [WIDTH_RESULT - 1..0] wide.
overflow	No	Overflow flag. If output_saturation is enabled, overflow flag is set.

Port Name	Required	Description
scanouta[]	No	Output of scan chain A. Output port [WIDTH_A - 1..0] wide. When designing with Stratix III devices, port cannot be selected when scaninb[] is in use. Do not use scanina[] and scaninb[] simultaneously.
scanoutb[]	No	Output of scan chain B. Output port [WIDTH_B - 1..0] wide. When designing with Stratix III devices, port cannot be selected when scanina[] is in use. Do not use scanina[] and scaninb[] simultaneously.

Table 9-6: ALTMULT_ADD Megafunction Output Ports (Stratix III and Stratix IV Devices Only)

Port Name	Required	Description
chainout_sat_overflow	No	Overflow flag for the chainout saturation.

ALTMULT_ADD Parameters

The following table lists the parameters for the ALTMULT_ADD megafunction.

Note: For Stratix III, Stratix IV, and Arria II GX devices, when the output result is > 36 bits (for example, when you set width_a=18 and width_b=18), the option for rounding and saturation is disabled. This is because additional logic is used to generate the MSB.

Table 9-7: ALTMULT_ADD Megafunction Parameters

Parameter Name	Type	Required	Description
NUMBER_OF_MULTIPLIERS	Integer	Yes	Number of multipliers to be added together. Values are 1 up to 4.
WIDTH_A	Integer	Yes	Width of the dataa[] port.
WIDTH_B	Integer	Yes	Width of the datab[] port.
WIDTH_RESULT	Integer	Yes	Width of the result[] port. Value includes all bits before rounding and saturation.
INPUT_REGISTER_A0	String	No	Specifies the clock port for the dataa[] operand of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, INPUT_REGISTER_A0 must have similar values with INPUT_REGISTER_A[1..3].

Parameter Name	Type	Required	Description
INPUT_REGISTER_A1	String	No	Specifies the clock port for the dataa[] operand of the second multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_A[1..3] must be set similar to the value of INPUT_REGISTER_A0.
INPUT_REGISTER_A2	String	No	Specifies the clock port for the dataa[] operand of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_A[1..3] must be set similar to the value of INPUT_REGISTER_A0.
INPUT_REGISTER_A3	String	No	Specifies the clock port for the dataa[] operand of the fourth and corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_A[1..3] must be set similar to the value of INPUT_REGISTER_A0.
INPUT_REGISTER_B0	String	No	Specifies the clock port for the datab[] operand of the first multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, INPUT_REGISTER_B0 must have similar values with INPUT_REGISTER_B[1..3].

Parameter Name	Type	Required	Description
INPUT_REGISTER_B1	String	No	Specifies the clock port for the datab[] operand of the second multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_B[1..3] must be set similar to the value of INPUT_REGISTER_B0.
INPUT_REGISTER_B2	String	No	Specifies the clock port for the datab[] operand of the third multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_B[1..3] must be set similar to the value of INPUT_REGISTER_B0.
INPUT_REGISTER_B3	String	No	Specifies the clock port for the datab[] operand of the fourth and corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0. For Stratix III devices, the values for INPUT_REGISTER_B[1..3] must be set similar to the value of INPUT_REGISTER_B0.
INPUT_ACLR_A0	String	No	Specifies the asynchronous clear for the dataa[] operand of the first multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_A[] is used, the default value is ACLR3.
INPUT_ACLR_A1	String	No	Specifies the asynchronous clear for the dataa[] operand of the second multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_A[] is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
INPUT_ACLR_A2	String	No	Specifies the asynchronous clear for the dataa[] operand of the third multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_A[] is used, the default value is ACLR3.
INPUT_ACLR_A3	String	No	Specifies the asynchronous clear for the dataa[] operand of the fourth and corresponding multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_A[] is used, the default value is ACLR3.
INPUT_ACLR_B0	String	No	Specifies the asynchronous clear for the datab[] operand of the first multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_B[] is used, the default value is ACLR3.
INPUT_ACLR_B1	String	No	Specifies the asynchronous clear for the datab[] operand of the second multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_B[] is used, the default value is ACLR3.
INPUT_ACLR_B2	String	No	Specifies the asynchronous clear for the datab[] operand of the third multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_B[] is used, the default value is ACLR3.
INPUT_ACLR_B3	String	No	Specifies the asynchronous clear for the datab[] operand of the fourth and corresponding multiplier. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding INPUT_REGISTER_B[] is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
INPUT_SOURCE_A0	String	No	Specifies the data source to the first multiplier. Values are DATAA and SCANA. If this parameter is set to DATAA, the adder uses the values from the dataa[] port. If this parameter is set to SCANA, the adder uses values from the scan chain. If omitted, the default value is DATAA. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier.
INPUT_SOURCE_A1	String	No	Specifies the data source to the second multiplier. Values are DATAA and SCANA. If this parameter is set to DATAA, the adder uses the values from the dataa[] port. If this parameter is set to SCANA, the adder uses values from the scan chain. If omitted, the default value is DATAA. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier.
INPUT_SOURCE_A2	String	No	Specifies the data source to the third multiplier. Values are DATAA and SCANA. If this parameter is set to DATAA, the adder uses the values from the dataa[] port. If this parameter is set to SCANA, the adder uses values from the scan chain. If omitted, the default value is DATAA. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier.

Parameter Name	Type	Required	Description
INPUT_SOURCE_A3	String	No	Specifies the data source to the fourth and corresponding multiplier. Values are DATAA and SCANA. If this parameter is set to DATAA, the adder uses the values from the dataa[] port. If this parameter is set to SCANA, the adder uses values from the scan chain. If omitted, the default value is DATAA. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier.
INPUT_SOURCE_B0	String	No	Specifies the data source of the first multiplier. Values are DATAB and SCANB. If this parameter is set to DATAB, then the adder uses the values from the datab[] port. If this parameter is set to SCANB, then the adder uses values from the scan chain. If omitted, the default value is DATAB. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier. For Stratix III devices in sum 2 (sum of two) mode, a value of LOOPBACK is available.
INPUT_SOURCE_B1	String	No	Specifies the data source of the second multiplier. Values are DATAB and SCANB. If this parameter is set to DATAB, then the adder uses the values from the datab[] port. If this parameter is set to SCANB, then the adder uses values from the scan chain. If omitted, the default value is DATAB. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier. For Stratix III devices in sum 2 (sum of two) mode, a value of LOOPBACK is available.

Parameter Name	Type	Required	Description
INPUT_SOURCE_B2	String	No	Specifies the data source of the third multiplier. Values are DATAB and SCANB. If this parameter is set to DATAB, then the adder uses the values from the datab[] port. If this parameter is set to SCANB, then the adder uses values from the scan chain. If omitted, the default value is DATAB. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier. For Stratix III in sum 2 (sum of two) mode, a value of LOOPBACK is available.
INPUT_SOURCE_B3	String	No	Specifies the data source of the fourth and corresponding multiplier. Values are DATAB and SCANB. If this parameter is set to DATAB, then the adder uses the values from the datab[] port. If this parameter is set to SCANB, then the adder uses values from the scan chain. If omitted, the default value is DATAB. For Stratix II devices, a value of VARIABLE is available for the adder to perform rounding and saturation on the data source before feeding the result to the multiplier. For Stratix III devices in sum 2 (sum of two) mode, a value of LOOPBACK is available.

Parameter Name	Type	Required	Description
REPRESENTATION_A	String	No	Specifies the numerical representation of the multiplier input A. Values are UNSIGNED, SIGNED and VARIABLE. When this parameter is set to SIGNED, the adder interprets the multiplier input A as a signed two's complement number. When this parameter is set to UNSIGNED, the adder interprets the multiplier input A as an unsigned number. If omitted, the default value is UNSIGNED. Use the VARIABLE setting to access the SIGNED_REGISTER_A and the SIGNED_PIPELINE_REGISTER_A parameter options for the signa input port.
REPRESENTATIONS_B	String	No	Specifies the numerical representation of the multiplier input B port. Values are UNSIGNED, SIGNED, and VARIABLE. When this parameter is set to UNSIGNED, the adder interprets the multiplier input B as an unsigned number. When this parameter is set to SIGNED, the adder interprets the multiplier input B as a signed two's complement number. If omitted, the default value is UNSIGNED. Use the VARIABLE setting to access the SIGNED_REGISTER_B and the SIGNED_PIPELINE_REGISTER_B parameter options for the signb input port.
SIGNED_REGISTER_[]	String	No	Parameter [A,B]. Specifies the clock signal for the first register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If the corresponding sign[] port value is UNUSED, this parameter is ignored. If omitted, the default value is CLOCK0.

Parameter Name	Type	Required	Description
SIGNED_PIPELINE_REGISTER_[]	String	No	Parameter [A,B]. Specifies the clock signal for the second register on the corresponding sign[] port. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If the corresponding sign[] port value is UNUSED, this parameter is ignored. If omitted, the default value is CLOCK0.
SIGNED_ACLR_[]	String	No	Parameter [A,B]. Specifies the asynchronous clear signal for the first register on the corresponding sign[] port. Values are NONE, ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding SIGNED_REGISTER_[] is used, the default value is ACLR3.
SIGNED_PIPELINE_ACLR_[]	String	No	Parameter [A,B]. Specifies the asynchronous clear signal for the second register on the corresponding sign[] port. Values are NONE,ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and the corresponding SIGNED_PIPELINE_REGISTER_[] is used, the default value is ACLR3.
MULTIPLIER_REGISTER[]	String	No	Parameter [0..3]. Specifies the clock source of the register that follows the corresponding multiplier. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
MULTIPLIER_ACLR[]	String	No	Parameter [0..3]. Specifies the asynchronous clear signal of the register that follows the corresponding multiplier. Values are NONE,ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and corresponding MULTIPLIER_REGISTER[] is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
MUTIPLIER1_DIRECTION	String	No	Specifies whether the second multiplier adds or subtracts its value from the sum. Values are ADD and SUB. If the addnsub1 port is used, this parameter is ignored. If omitted, the default value is ADD.
MUTIPLIER3_DIRECTION	String	No	Specifies whether the fourth and all subsequent odd-numbered multipliers add or subtract their results from the total. Values are ADD and SUB. If the addnsub3 port is used, this parameter is ignored. If omitted, the default value is ADD.
ACCUM_DIRECTION	String	No	Specifies whether to use the accumulator and whether the accumulator adds or subtracts its value from the sum. Values are ADD and SUB. If omitted, the default value is ADD.
OUTPUT_REGISTER	String	No	Specifies the clock signal for the second adder register. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
OUTPUT_ACLR	String	No	Specifies the asynchronous clear signal for the second adder register. Values are NONE, ACLR0, ACLR1, ACLR2, and ACLR3. If omitted, the default value is ACLR3.
PORT_SIGN[]	String	No	Parameter [A,B]. Specifies the corresponding sign[] input port usage. Values are PORT_USED, PORT_UNUSED, and PORT_CONNECTIVITY. If omitted, the default value is PORT_CONNECTIVITY.
CHAINOUT_ROUND_TYPE	String	No	Specifies the rounding mode at the chainout stage. Values are BIASED and UNBIASED. A value of BIASED specifies round-to-nearest-integer. A value of UNBIASED specifies round-to-nearest-even.

Parameter Name	Type	Required	Description
EXTRA_LATENCY	String	No	Specifies the number of clock cycles of latency.
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES" The default value is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the ALTMULT_ADD megafunction with the MegaWizard Plug-in Manager to calculate the value for this parameter.
DSP_BLOCK_BALANCING	String	No	If omitted, the default value is AUTO.
DEDICATED_MULTIPLIER_CIRCUITRY	String	No	Specifies whether to use the DSP block to implement the circuit. Values are YES, NO, and AUTO. The circuit is implemented using the DSP block when the value is set to YES. If omitted, the default value is AUTO.

Table 9-8: ALTMULT_ADD Megafunction Parameters (Stratix II, Stratix III, and Stratix IV Devices Only)

Parameter Name	Type	Required	Description
OUTPUT_SATURATE_TYPE	String	No	Specifies the saturation mode. Values are SYMMETRIC and ASYMMETRIC. A value of SYMMETRIC specifies the absolute value of the maximum negative number equal to the maximum positive number. A value of ASYMMETRIC specifies the maximum negative number is larger than the maximum positive number. If omitted, the default value is ASYMMETRIC.
WIDTH_SATURATE_SIGN	String	No	Specifies the saturation position. The value is determined by counting the bits that become the sign bits after saturation. Values are calculated according to the following modes- WIDTH_A, WIDTH_B, and WIDTH_RESULT. Value must be an unsigned integer. If a positive number is unavailable, no saturation is allowed in your input/output width and mode setting. If omitted, the default value is 1.
CHAINOUT_ADDER	String	No	Specifies the chainout mode of the final adder stage. Values are YES and NO. If omitted, the default value is NO.
ACCUMULATOR	String	No	Specifies the accumulator mode of the final adder stage. Values are YES and NO. If omitted, the default value is NO. When value is set to YES, rounding is dynamic and you must initialize the accumulator while rounded data is acquired.

Table 9-9: ALTMULT_ADD Megafunction Parameters (Stratix III and Stratix IV Devices Only)

Parameter Name	Type	Required	Description
WIDTH_CHAININ	Integer	No	Width of the chainin[] port. WIDTH_CHAININ equals WIDTH_RESULT if port chainin is used. If omitted, the default value is 1.

Parameter Name	Type	Required	Description
OUTPUT_ROUNDING	String	No	Enables rounding handling at second adder stage. If original design uses a Stratix II device, in some cases this parameter can be derived from the Stratix II rounding settings. Values are YES, NO, and VARIABLE. A value of YES or NO specifies saturation handling setting permanently to on or off. A value of VARIABLE allows dynamically controlled saturation handling.
OUTPUT_ROUND_TYPE	String	No	Specifies the rounding mode. Values are NEAREST_EVEN and NEAREST_INTEGER. A value of NEAREST_EVEN specifies round-to-nearest-even. A value of NEAREST_INTEGER specifies round-to-nearest-integer. If omitted, the default value is NEAREST_INTEGER.
OUTPUT_ROUND_REGISTER	String	No	Specifies the clock source for the first register on the output_round input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
OUTPUT_ROUND_ACLR	String	No	Specifies the asynchronous clear source for the first register on the output_round input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and OUTPUT_ROUND_REGISTER is used, the default value is ACLR3.
OUTPUT_ROUND_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the output_round input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
OUTPUT_ROUND_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the output_round input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and OUTPUT_ROUND_PIPELINE_ REGISTER is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
OUTPUT_SATURATION	String	No	Enables saturation handling at second adder stage. If original design uses a Stratix II device, in some cases this parameter can be derived from the Stratix II rounding settings. Values are YES, NO, and VARIABLE. A value of YES or NO specifies saturation handling setting permanently to on or off. A value of VARIABLE allows dynamically controlled saturation handling. If omitted, the default value is NO.
OUTPUT_SATURATE_REGISTER	String	No	Specifies the clock source for the first register on the output_saturate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is UNREGISTERED.
OUTPUT_SATURATE_ACLR	String	No	Specifies the asynchronous clear source for the first register on the output_saturate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and OUTPUT_SATURATE_REGISTER is used, the default value is ACLR3.
OUTPUT_SATURATE_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the output_saturate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
OUTPUT_SATURATE_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the output_saturate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and OUTPUT_SATURATE_ PIPELINE_REGISTER is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
CHAINOUT_ROUNDING	String	No	Enables rounding handling at the chainout stage. Values are YES, NO, and VARIABLE. A value of YES or NO specifies saturation handling setting permanently to on or off. A value of VARIABLE allows dynamically controlled saturation handling. If the value of CHAINOUT_ROUNDING is YES, the symmetric saturation at the second adder output stage is not allowed. If omitted, the default value is NO.
CHAINOUT_ROUND_REGISTER	String	No	Specifies the clock source for the first register on the chainout_round input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
CHAINOUT_ROUND_ACLR	String	No	Specifies the asynchronous clear source for the first register on the chainout_round input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_ROUND_REGISTER is used, the default value is ACLR3.
CHAINOUT_ROUND_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the chainout_round input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
CHAINOUT_ROUND_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the chainout_round input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_ROUND_ PIPELINE_REGISTER is used, the default value is ACLR3.
CHAINOUT_ROUND_OUTPUT_REGISTER	String	No	Specifies the clock source for the third register on the chainout_round input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.

Parameter Name	Type	Required	Description
CHAINOUT_ROUND_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the third register on the chainout_round input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_ROUND_ OUTPUT_ REGISTER is used, the default value is ACLR3.
CHAINOUT_SATURATION	String	No	Enables saturation handling at the chainout stage. Values are YES, NO, and VARIABLE. A value of YES or NO specifies saturation handling setting permanently to on or off. A value of VARIABLE allows dynamically controlled saturation handling. If omitted, the default value is NO.
CHAINOUT_SATURATE_REGISTER	String	No	Specifies the clock source for the first register on the chainout_ saturate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
CHAINOUT_SATURATE_ACLR	String	No	Specifies the asynchronous clear source for the first register on the chainout_saturate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_SATURATE_REGISTER is used, the default value is ACLR3.
CHAINOUT_SATURATE_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the chainout_ saturate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
CHAINOUT_SATURATE_OUTPUT_REGISTER	String	No	Specifies the clock source for the third register on the chainout_ saturate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.

Parameter Name	Type	Required	Description
CHAINOUT_SATURATE_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the third register on the chainout_saturate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_SATURATE_ OUTPUT_ REGISTER is used, the default value is ACLR3.
ZERO_CHAINOUT_OUTPUT_REGISTER	String	No	Specifies the clock source for the first register on the zero_ chainout input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ZERO_CHAINOUT_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the first register on the zero_chainout input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ZERO_CHAINOUT_ OUTPUT_ REGISTER is used, the default value is ACLR3.
ZERO_LOOPBACK_REGISTER	String	No	Specifies the clock source for the first register on the zero_ loopback input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ZERO_LOOPBACK_ACLR	String	No	Specifies the asynchronous clear source for the first register on the zero_loopback input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ZERO_LOOPBACK_ PIPELINE_REGISTER is used, the default value is ACLR3.
ZERO_LOOPBACK_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the zero_ loopback input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ZERO_LOOPBACK_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the zero_loopback input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ZERO_ LOOPBACK_ PIPELINE_REGISTER is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
ZERO_LOOPBACK_OUTPUT_REGISTER	String	No	Specifies the clock source for the third register on the zero_loopback input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ZERO_LOOPBACK_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the third register on the zero_loopback input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ZERO_LOOPBACK_OUTPUT_ REGISTER is used, the default value is ACLR3.
ACCUM_SLOAD_REGISTER	String	No	Specifies the clock source for the first register on the accum_sload input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ACCUM_SLOAD_ACLR	String	No	Specifies the asynchronous clear source for the first register on the accum_sload input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ACCUM_SLOAD_REGISTER is used, the default value is ACLR3.
ACCUM_SLOAD_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the accum_sload input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ACCUM_SLOAD_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the accum_sload input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ACCUM_SLOAD_PIPELINE_ REGISTER is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
SHIFT_MODE	String	No	Specifies the shift mode. Values are NO, LEFT, RIGHT, ROTATION, and VARIABLE. If VARIABLE is selected, rotate and shift_right are used to specify shift left, shift right, or rotation. If omitted, the default value is NO. Note that this parameter is supported only when inputs equal 32 bits each, output equals 32 bits, and the number of multipliers equals 1.
ROTATE_REGISTER	String	No	Specifies the clock source for the first register on the rotate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ROTATE_ACLR	String	No	Specifies the asynchronous clear source for the first register on the rotate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ROTATE_REGISTER is used, the default value is ACLR3.
ROTATE_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the rotate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
ROTATE_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the rotate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ROTATE_PIPELINE_REGISTER is used, the default value is ACLR3.
ROTATE_OUTPUT_REGISTER	String	No	Specifies the clock source for the third register on the rotate input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.

Parameter Name	Type	Required	Description
ROTATE_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the third register on the rotate input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and ROTATE_OUTPUT_REGISTER is used, the default value is ACLR3.
SHIFT_RIGHT_REGISTER	String	No	Specifies the clock source for the first register on the shift_right input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
SHIFT_RIGHT_ACLR	String	No	Specifies the asynchronous clear source for the first register on the shift_right input. Values are NONE, ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and SHIFT_RIGHT_REGISTER is used, the default value is ACLR3.
SHIFT_RIGHT_PIPELINE_REGISTER	String	No	Specifies the clock source for the second register on the shift_right input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
SHIFT_RIGHT_PIPELINE_ACLR	String	No	Specifies the asynchronous clear source for the second register on the shift_right input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and SHIFT_RIGHT_PIPELINE_ REGISTER is used, the default value is ACLR3.
SHIFT_RIGHT_OUTPUT_REGISTER	String	No	Specifies the clock source for the third register on the shift_right input. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
SHIFT_RIGHT_OUTPUT_ACLR	String	No	Specifies the asynchronous clear source for the third register on the shift_right input. Values are ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and SHIFT_RIGHT_OUTPUT_REGISTER is used, the default value is ACLR3.

Parameter Name	Type	Required	Description
PORT_OUTPUT_IS_OVERFLOW	String	No	Specifies port usage. Values are PORT_UNUSED and PORT_USED. When the value is set to PORT_USED, output pin overflow is added. If omitted, the default value is PORT_UNUSED.
PORT_CHAINOUT_SAT_IS_OVERFLOW	String	No	Specifies port usage. Values are PORT_UNUSED and PORT_USED. When the value is set to PORT_USED, output pin chainout_sat_overflow is added. If omitted, the default value is PORT_UNUSED.
SCANOUTA_REGISTER	String	No	Specifies the clock source for the scanouta data bus registers. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is UNREGISTERED.
SCANOUTA_ACLR	String	No	Specifies the asynchronous clear source for the scanouta data bus registers. Values are NONE, ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and SCANOUTA_REGISTER is used, the default value is ACLR3.
CHAINOUT_REGISTER	String	No	Specifies the clock source for the chainout mode result register. This is an additional stage after the second adder. Values are UNREGISTERED, CLOCK0, CLOCK1, CLOCK2, and CLOCK3. If omitted, the default value is CLOCK0.
CHAINOUT_ACLR	String	No	Specifies the asynchronous clear for the chainout mode result register. This is an additional stage after the second adder. Values are NONE, ACLR0, ACLR1, ACLR2, and ACLR3. If omitted and CHAINOUT_REGISTER is used, the default value is ACLR3.

Design Example: Implementing a Simple Finite Impulse Response (FIR) Filter

This design example uses the ALTMULT_ADD megafunction to implement a simple FIR filter as shown in the following equation. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

$$y(t) = \sum_{i=0}^{n-1} A(t-i)B(i)$$

n represents the number of taps, *A(t)* represents the sequence of input samples, and *B(i)* represents the filter coefficients.

The number of taps *(n)* can be any value, but this example is of a simple FIR filter with *n* = 4, which is called a 4-tap filter. To implement this filter, the coefficients of data *B* is loaded into the *B* registers in parallel and a `shiftin` register moves data *A(0)* to *A(1)* to *A(2)*, and so on. With a 4-tap filter, at a given time *(t)*, the sum of four products is computed. This function is implemented using the shift register chain option in the ALTMULT_ADD megafunction.

With reference to the equation, input *B* represents the coefficients and data *A* represents the data that is shifted into. The *A* input (data) is shifted in with the main clock, named `clock0`. The *B* input (coefficients) is loaded at the rising edge of `clock1` with the enable signal held high.

The following design files can be found in **altmult_add_DesignExample.zip**:

fir_fourtap.qar (archived Quartus II design files)

altmult_add_ex_msim (ModelSim-Altera files)

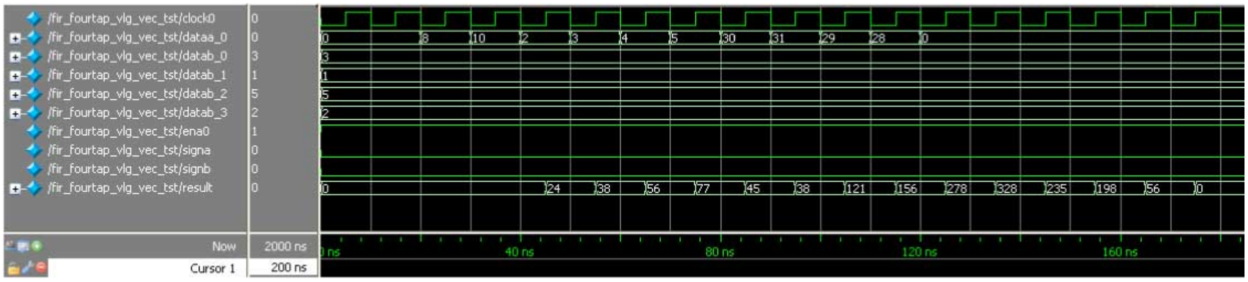
Understanding the Simulation Results

The following settings are observed in this example:

- The widths of the data inputs are all set to 16 bits
- The width of the output port, `result[]`, is set to 34 bits
- The input registers are all operating on the same clock

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 9-13: ALTMULT_ADD Simulation Results



ALTMULT_COMPLEX (Complex Multiplier) 10

2014.12.19

UG-01063 Subscribe Send Feedback

The ALTMULT_COMPLEX megafunction implements the multiplication of two complex numbers and offers the conventional implementation mode.

You can use the conventional representation for the following:

- All supported Altera devices
- Input data widths of any size

The ALTMULT_COMPLEX megafunction implements the multiplication of two complex numbers and offers the following two implementation modes:

- Canonical

  You can use the canonical representation for the following:

  - All supported Altera devices prior to Stratix III devices. The canonical representation is no longer supported from Stratix III onwards
  - Input data widths of less than 18 bits
- Conventional

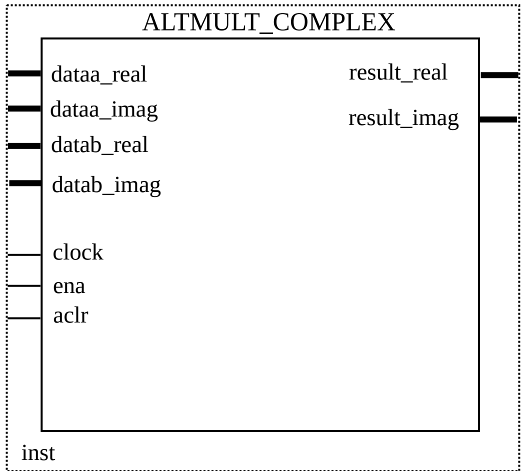
  You can use the conventional representation for the following:

  - All supported Altera devices
  - Input data widths of any size

With the conventional representation, you can use the ALTMULT_ADD megafunction to implement the complex multiplier by instantiating two multipliers.

The following figure shows the ports for the ALTMULT_COMPLEX megafunction.

Figure 10-1: ALTMULT_COMPLEX Ports



Complex Multiplication

Complex numbers are numbers in the form of the following equation:

$a + ib$

Where:

- a and b are real numbers
- i is an imaginary unit that equals the square root of -1:$\sqrt{-1}$

Two complex numbers, $x = a + ib$ and $y = c + id$ are multiplied, as shown in the following equations.

$$xy = (a + ib)(c + id)$$
$$= ac + ibc + iad - bd$$

$$= (ac - bd) + i(ad + bc)$$

Related Information

Canonical Representation on page 10-2

Canonical Representation

From Complex Multiplication equation, the multiplication of two complex numbers can be represented in two parts: real and imaginary.

The following equation shows that the *xy_real* variable represents real representation.

$$\begin{aligned} xy\_real &= ac - bd \\ &= ac - bd + (ad - bc) - (ad - bc) \\ &= (ac - ad + bc - bd) + (ad - bc) \\ &= ((a + b)(c - d)) + (ad - bc) \end{aligned}$$

xy_real represents the real part

The following equation shows that the *xy_imaginary* variable represents imaginary representation.

$$xy\_imaginary = ad + bc$$

xy_imaginary represents imaginary

Both equations derived from Complex Multiplication equation.

Note: The canonical representation is available for all supported Altera devices prior to Stratix III devices.

Related Information

Complex Multiplication on page 10-2

Conventional Representation

The multiplication of two complex numbers can be represented in two parts, real and imaginary. The *xy_real* variable in the following equation represents the real part:

$xy\_real = ac - bd$

The *xy_imaginary* variable in the following equation represents the imaginary part.

$xy\_imaginary = ad + bc$

Features

The ALTMULT_COMPLEX megafunction offers the following features:

- Generates a multiplier to perform multiplication operations of two complex numbers

  Note: When building multipliers larger than the natively supported size there may/will be a performance impact resulting from the cascading of the DSP blocks.
- Supports data width of 1–256 bits
- Supports signed and unsigned data representation format
- Supports canonical and conventional implementation modes
- Supports pipelining with configurable output latency
- Supports optional asynchronous clear and clock enable input ports
- Provides an option to dynamically switch between 36 × 36 normal mode and 18 × 18 complex mode (for Stratix V devices only)

Resource Utilization and Performance

The following table provides resource utilization and performance information for the ALTMULT_COMPLEX megafunction.

Table 10-1: ALTMULT_COMPLEX Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			18-bit DSP	f_{MAX} (MHz) [7]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)		
Stratix III	8	0	0	0	0	4	529
	16	0	0	0	0	4	531
	32	0	73	0	37	16	291
	64	0	73	0	37	16	292
	8	14	19	10	10	4	492
	16	14	19	10	10	4	502
	32	14	91	8	47	16	265
	64	14	91	8	47	16	268
Stratix IV	8	0	0	0	0	4	487
	16	0	0	0	0	4	487
	32	0	73	0	37	16	293
	64	0	73	0	37	16	293
	8	14	19	10	10	4	489
	16	14	20	10	10	4	493
	32	14	91	8	47	16	292
	64	14	91	8	47	16	291

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **altera_mf.v** in the *<Quartus II installation directory>***\eda\synthesis** directory.

```
module altmult_complex
# (parameter intended_device_family = "unused",
parameter implementation_style = "AUTO",
parameter pipeline = 4,
```

(7) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

```
parameter representation_a = "SIGNED",
parameter representation_b = "SIGNED",
parameter width_a = 1,
parameter width_b = 1,
parameter width_result = 1,
parameter lpm_type = "altmult_complex",
parameter lpm_hint = "unused")
(input wire aclr,
input wire clock,
input wire complex,
input wire [width_a-1:0] dataa_imag,
input wire [width_a-1:0] dataa_real,
input wire [width_b-1:0] datab_imag,
input wire [width_b-1:0] datab_real,
input wire ena,
output wire [width_result-1:0] result_imag,
output wire [width_result-1:0] result_real;
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altmult_complex
generic (
intended_device_family:string := "unused";
implementation_style:string := "AUTO";
pipeline:natural := 4;
representation_a:string := "SIGNED";
representation_b:string := "SIGNED";
width_a:natural;
width_b:natural;
width_result:natural;
lpm_hint:string := "UNUSED";
lpm_type:string := "altmult_complex");
port(
aclr:in std_logic := '0';
clock:in std_logic := '0';
complex:in std_logic := '1';
dataa_imag:in std_logic_vector(width_a-1 downto 0);
dataa_real:in std_logic_vector(width_a-1 downto 0);
datab_imag:in std_logic_vector(width_b-1 downto 0);
datab_real:in std_logic_vector(width_b-1 downto 0);
ena:in std_logic := '1';
result_imag:out std_logic_vector(width_result-1 downto 0);
result_real:out std_logic_vector(width_result-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

ALTMULT_COMPLEX Ports

Table 10-2: ALTMULT_COMPLEX IP Core Input Ports

Port Name	Required	Description
aclr	No	Asynchronous clear for the complex multiplier. When the aclr port is asserted high, the function is asynchronously cleared.
clock	Yes	Clock input to the ALTMULT_COMPLEX function.
dataa_imag[]	Yes	Imaginary input value for the data A port of the complex multiplier. The size of the input port depends on the WIDTH_A parameter value.
dataa_real[]	Yes	Real input value for the data A port of the complex multiplier. The size of the input port depends on the WIDTH_A parameter value.
datab_imag[]	Yes	Imaginary input value for the data B port of the complex multiplier. The size of the input port depends on the WIDTH_B parameter value.
datab_real[]	Yes	Real input value for the data B port of the complex multiplier. The size of the input port depends on the WIDTH_B parameter value.
ena	No	Active high clock enable for the clock port of the complex multiplier.

Table 10-3: ALTMULT_COMPLEX IP Core Output Ports

Port Name	Required	Description
result_imag	Yes	Imaginary output value of the multiplier. The size of the output port depends on the WIDTH_RESULT parameter value.
result_real	Yes	Real output value of the multiplier. The size of the output port depends on the WIDTH_RESULT parameter value.

ALTMULT_COMPLEX Parameters

The following table lists the parameters for the ALTMULT_COMPLEX megafunction.

Table 10-4: ALTMULT_COMPLEX Megafunction Parameters

Parameter Name	Type	Required	Description
IMPLEMENTATION_STYLE	String	Yes	Specifies the representation algorithm and the number of bits per channel. Values are AUTO, CANONICAL, and CONVENTIONAL. If omitted, the default value is AUTO. When set to AUTO, the Quartus II software determines the best implementation based on the selected device family and input width. A value of CANONICAL is available for input widths that are less than 18 bits and for all supported devices, except for Stratix III devices. A value of CONVENTIONAL is available for all supported device families for all input ranges (1 to 256 bits).
PIPELINE	Integer	Yes	Specifies the amount of latency, in clock cycles, needed to produce the result. Values are [0..14]. If omitted, the default value is 4. If the value of IMPLEMENTATION_STYLE is CANONICAL, the maximum value of PIPELINE is 14, and if the value of the IMPLEMENTATION_STYLE parameter is CONVENTIONAL, the maximum value of PIPELINE is 11.
REPRESENTATION_A	String	Yes	Specifies the number representation of data A. Values are UNSIGNED and SIGNED. If omitted, the default value is UNSIGNED. The data A inputs are interpreted as unsigned numbers when the value is set to UNSIGNED, and as two's complement when the value is set to SIGNED.
REPRESENTATION_B	String	Yes	Specifies the number representation of data B. Values are UNSIGNED and SIGNED. If omitted, the default value is UNSIGNED. The data B inputs are interpreted as unsigned numbers when the value is set to UNSIGNED, and as two's complement when the value is set to SIGNED.
WIDTH_A	Integer	Yes	Specifies the width of the dataa_real[] and dataa_imag[] ports. Value must be 256 bits or less. If omitted, the default value is 18.

Parameter Name	Type	Required	Description
`WIDTH_B`	Integer	Yes	Specifies the width of the `datab_real[]` and `datab_imag[]` ports. Value must be `256` bits or less. If omitted, the default value is `18`.
`WIDTH_RESULT`	Integer	Yes	Specifies the width of the `result_real[]` and `result_imag[]` ports. Value must be `256` bits or less. If omitted, the default value is `36`.
`INTENDED_DEVICE_FAMILY`	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the `ALTMULT_COMPLEX` megafunction with the MegaWizard Plug-in Manager to calculate the value for this parameter.
`LPM_HINT`	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the `LPM_HINT` parameter to specify an Altera-specific parameter. For example: `LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES"` The default value is `UNUSED`.
`LPM_TYPE`	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.

Design Example: Multiplication of 8-bit Complex Numbers Using Canonical Representation

This design example uses the ALTMULT_COMPLEX megafunction to implement a complex multiplier with an 8-bit input data width using the canonical representation. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **altmult_complex_DesignExample.zip**:

complex_canonical.qar (archived Quartus II design files)

altmult_complex_ex_msim (ModelSim-Altera files)

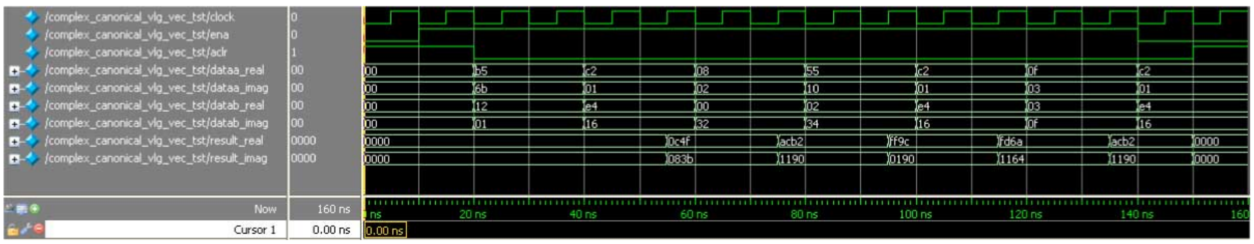
Understanding the Simulation Results

The following settings are observed in this example:

- The widths of the data inputs are all set to 8 bits
- The widths of the output ports are set to 16 bits
- The asynchronous clear (`aclr`) and clock enable (`ena`) signals are enabled
- Pipelining is enabled with an output latency of four clock cycles. Hence, the result is seen on the output ports four clock cycles after the input data is available

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 10-2: ALTMULT_COMPLEX Simulation Results



ALTSQRT (Integer Square Root) 11

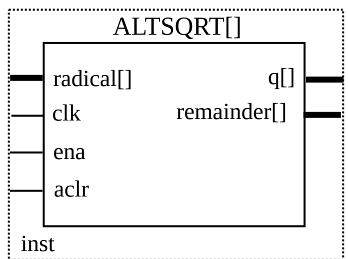
2014.12.19

UG-01063 Subscribe Send Feedback

The ALTSQRT megafunction implements a square root function that calculates the square root and remainder of an input.

The following figure shows the ports for the ALTSQRT megafunction.

Figure 11-1: ALTSQRT Ports



Features

The ALTSQRT megafunction offers the following features:

- Calculates the square root and the remainder of an input
- Supports data width of 1–256 bits
- Supports pipelining with configurable output latency
- Supports optional asynchronous clear and clock enable input ports

Resource Utilization and Performance

The following table provides resource utilization and performance information for the ALTSQRT megafunction.

Device family	Input data width	Output latency	Logic Usage			f_{MAX} (MHz) [8]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
Stratix III	8	1	28	0	14	547
	20	5	131	0	90	321
	30	3	256	0	152	71
Stratix IV	8	1	28	0	14	350
	20	5	131	0	94	267
	30	3	256	0	154	66

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **altera_mf.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module altsqrt
# (parameter lpm_hint = "UNUSED",
parameter lpm_type = "altsqrt",
parameter pipeline = 0,
parameter q_port_width = 1,
parameter r_port_width = 1,
parameter width = 1)
(input wire aclr,
input wire clk,
input wire ena,
output wire [q_port_width-1:0] q,
input wire [width-1:0] radical,
output wire [r_port_width-1:0] remainder);
endmodule
```

VHDL Component Declaration

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component altsqrt
generic (
lpm_hint:string := "UNUSED";
lpm_type:string := "altsqrt";
pipeline:natural := 0;
q_port_width:natural := 1;
r_port_width:natural := 1;
width:natural);
port(
aclr:in std_logic := '0';
```

(8) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

```
clk:in std_logic := '1';
ena:in std_logic := '1';
q:out std_logic_vector(Q_PORT_WIDTH-1 downto 0);
radical:in std_logic_vector(WIDTH-1 downto 0);
remainder:out std_logic_vector(R_PORT_WIDTH-1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

Ports

The following tables list the input and output ports for the ALTSQRT megafunction.

Table 11-1: ALTSQRT Megafunction Input Ports

Port Name	Required	Description
`radical[]`	Yes	Data input port. The size of the input port depends on the `WIDTH` parameter value.
`ena`	No	Active high clock enable input port.
`clk`	No	Clock input port that provides pipelined operation for the `ALTSQRT` megafunction. For the values of `PIPELINE` parameter other than `0` (default value), the clock port must be connected.
`aclr`	No	Asynchronous clear input port. that can be used at any time to reset the pipeline to all `0`s, asynchronously to the clock signal.

Table 11-2: ALTSQRT Megafunction Output Ports

Port Name	Required	Description
`remainder[]`	Yes	The square root of the radical. The size of the `remainder[]` port depends on the `R_PORT_WIDTH` parameter value.
`q[]`	Yes	Data output. The size of the `q[]` port depends on the `Q_PORT_WIDTH` parameter value.

Parameters

The following table lists the parameters for the ALTSQRT megafunction.

Parameter Name	Type	Required	Description
`WIDTH`	Integer	Yes	Specifies the widths of the `radical[]` input port.
`Q_PORT_WIDTH`	Integer	Yes	Specifies the width of the `q[]` output port.

Parameter Name	Type	Required	Description
R_PORT_WIDTH	Integer	Yes	Specifies the width of the `remainder[]` output port.
PIPELINE	Integer	No	Specifies the number of clock cycles of latency to add.
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the `LPM_HINT` parameter to specify an Altera-specific parameter. For example: `LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES"` The default value is `UNUSED`.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.

Design Example: 9-bit Square Root

This design example uses the ALTSQRT megafunction to generate a 9-bit square root. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **altsqrt_DesignExample.zip**:

altsqrt.qar (archived Quartus II design files)

altsqrt_ex_msim (ModelSim-Altera files)

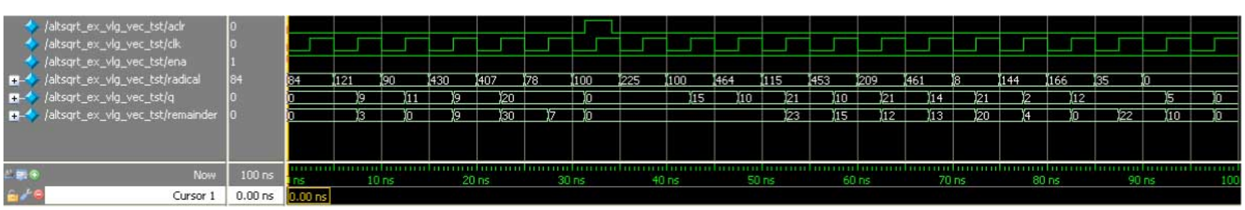
Understanding the Simulation Results

The following settings are observed in this example:

- The width of the input port, radical[], is set to 9 bits.
- The widths of the output ports, q[] and remainder[], are set to 5 bits and 6 bits respectively.
- The asynchronous clear (aclr) and clock enable (ena) input ports are enabled.
- The output latency is set to two clock cycles. Hence, the result is seen on the q[] port two clock cycles after the input data is available.

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 11-2: ALTSQRT Simulation Results



PARALLEL_ADD (Parallel Adder) 12

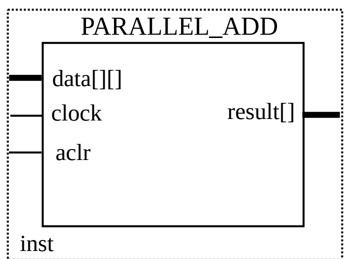
2014.12.19

UG-01063 Subscribe Send Feedback

The PARALLEL_ADD megafunction performs add or subtract operations on a selected number of inputs to produce a single sum result. You can add or subtract more than two operands and automatically shift the input operands upon entering the function. The method of shifting input operands is useful for serial FIR filter structures requiring a shift-and-accumulate of the partial products.

The following figure shows the ports for the PARALLEL_ADD megafunction.

Figure 12-1: PARALLEL_ADD Ports



Feature

The PARALLEL_ADD megafunction offers the following features:

- Performs add or subtract operations on a number of inputs to produce a single sum result
- Supports data width of 8–128 bits
- Supports signed and unsigned data representation format
- Supports pipelining with configurable output latency
- Supports shifting data vectors
- Supports addition or subtraction of the most-significant input operands
- Supports optional asynchronous clear and clock enable ports

Resource Utilization and Performance

The following table provides resource utilization and performance information for the PARALLEL_ADD megafunction.

Table 12-1: PARALLEL_ADD Resource Utilization and Performance

Device family	Input data width	Output latency	Logic Usage			f_{MAX} (MHz) [9]
			Adaptive Look-Up Table (ALUT)	Dedicated Logic Register (DLR)	Adaptive Logic Module (ALM)	
Stratix III	8	1	40	0	21	793
	32	5	142	0	102	378
	64	10	283	0	189	280
Stratix IV	8	1	40	0	21	854
	32	5	142	0	103	472
	64	10	283	0	199	346

Verilog HDL Prototype

The following Verilog HDL prototype is located in the Verilog Design File (**.v**) **altera_mf.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

```
module parallel_add (
    data,
    clock,
    aclr,
    clken,
    result);
    parameter width = 4;
    parameter size = 2;
    parameter widthr = 4;
    parameter shift = 0;
    parameter msw_subtract = "NO";  // or "YES"
    parameter representation = "UNSIGNED";
    parameter pipeline = 0;
    parameter result_alignment = "LSB"; // or "MSB"
    parameter lpm_type = "parallel_add";
    input [width*size-1:0] data;
    input clock;
    input aclr;
    input clken;
    output [widthr-1:0] result;
endmodule
```

VHDL Component Declaration

(9) The performance of the megafunction is dependant on the value of the maximum allowable ceiling f_{MAX} that the selected device can achieve. Therefore, results may vary from the numbers stated in this column.

The VHDL component declaration is located in the VHDL Design File (**.vhd**) **altera_mf_components.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

```
component parallel_add
   generic (
      width : natural := 4;
      size : natural := 2;
      widthr : natural := 4;
      shift : natural := 0;
      msw_subtract : string  := "NO";
      representation : string  := "UNSIGNED";
      pipeline : natural := 0;
      result_alignment : string  := "LSB";
      lpm_hint: string  := "UNUSED";
      lpm_type : string  := "parallel_add");
   port (
      data:in altera_mf_logic_2D(size - 1 downto 0,width- 1 downto 0);
      clock : in std_logic := '1';
      aclr : in std_logic := '0';
      clken : in std_logic := '1';
      result : out std_logic_vector(widthr - 1 downto 0));
end component;
```

VHDL LIBRARY_USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL Component Declaration.

```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

Ports

The following tables list the input and output ports of the PARALLEL_ADD megafunction.

Table 12-2: PARALLEL_ADD Megafunction Input Ports

Port Name	Required	Description
`data[]`	Yes	Data input to the parallel adder. Input port `[SIZE - 1 DOWNTO 0, WIDTH-1 DOWNTO 0]` wide.
`clock`	No	Clock input to the parallel adder. This port is required if the `PIPELINE` parameter has a value of greater than `0`.
`clken`	No	Clock enable to the parallel adder. If omitted, the default value is `1`.
`aclr`	No	Active high asynchronous clear input to the parallel adder.

Table 12-3: PARALLEL_ADD Megafunction Output Ports

Port Name	Required	Description
`result[]`	Yes	Adder output port. The size of the output port depends on the `WIDTHR` parameter value.

Parameters

The following table lists the parameters for the PARALLEL_ADD megafunction.

Table 12-4: PARALLEL_ADD Megafunction Parameters

Parameter Name	Type	Required	Description
WIDTH	Integer	Yes	Specifies the width of the data[] input port.
SIZE	Integer	Yes	Specifies the number of inputs to add.
WIDTHR	Integer	Yes	Specifies the width of the result[] output port.
SHIFT	Integer	Yes	Specifies the relative shift of the data vectors.
NEW_SUBTRACT	String	No	Specifies whether to add or subtract the most significant input word bit. Values are NO or YES. If omitted, the default value is NO.
REPRESENTATION	String	No	Specifies whether the input is signed or unsigned. Values are UNSIGNED or SIGNED. If omitted, the default value is UNSIGNED.
PIPELINE	Integer	No	Specifies the value, in clock cycles, of the output latency.
RESULT_ALIGNMENT	String	No	Specifies the alignment of the result port. Values are MSB or LSB. If omitted, the default value is LSB.
INTENDED_DEVICE_FAMILY	String	No	This parameter is used for modeling and behavioral simulation purposes. Create the ALTCDR_RX megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.
LPM_HINT	String	No	When you instantiate a library of parameterized modules (LPM) function in a VHDL Design File (**.vhd**), you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: LPM_HINT = "CHAIN_SIZE = 8, ONE_INPUT_IS_CONSTANT = YES" The default value is UNUSED.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL design files.

Design Example: Shift Accumulator

This design example uses the LPM_MULT and PARALLEL_ADD megafunctions to generate a shift accumulator. This function implements the shift-and-accumulate operation after the multiplication process in a design block, such as a serial FIR filter. This example uses the MegaWizard Plug-In Manager in the Quartus II software.

The following design files can be found in **parallel_adder_DesignExample.zip**:

- **shift_accum.qar** (archived Quartus II design files)
- **parallel_adder_ex_msim** (ModelSim-Altera files)

Understanding the Simulation Results

The following settings are observed in this example:

The widths of the input ports, `dataa[]` and `datab[]`, are set to 9 bits

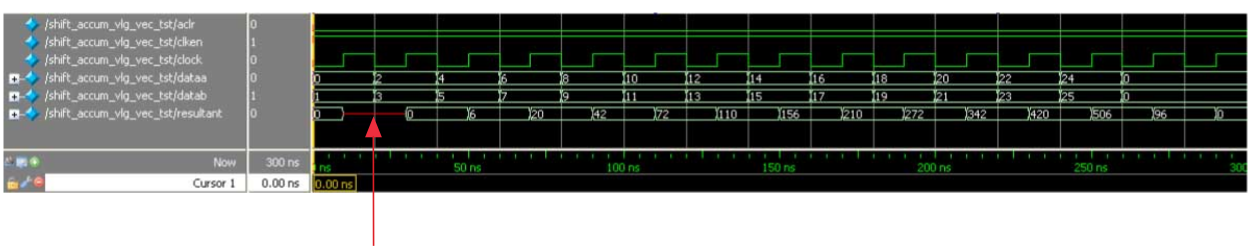
The width of the output port, `resultant[]`, is set to 10 bits

The asynchronous clear (`aclr`) and clock enable (`clocken`) input ports are enabled

The latency is set to one clock cycle for the multiplier and one clock cycle for the parallel adder, resulting in a total output latency of two clock cycles. Hence, the result is seen on the `resultant[]` port two clock cycles after the input data is available.

The following figure shows the expected simulation results in the ModelSim-Altera software.

Figure 12-2: PARALLEL_ADDER Simulation Results



Note: At start up, an undefined value is seen on the `resultant[]` port, but this value is merely due to the behavior of the system during start-up and hence, can be ignored

Document Revision History 13

2014.12.19

UG-01063 Subscribe Send Feedback

The following table lists the revision history for this document.

Table 13-1: Document Revision History

Date	Version	Changes
December, 2014	2014.12.19	• Removed the LPM_ADD_SUB and LPM_COMPARE IPs because these IPs are no longer supported. • Added a note to clarify that when building multipliers larger than the natively supported size there may be a performance impact resulting from the cascading of the DSP blocks in LPM_MULT, ALTERA_MULT_ADD, ALTMULT_ACCUM, ALTMULT_ADD, and ALTMULT_COMPLEX IP cores. • Added information about the **Create a 'sync_e' port** parameter and the `sync_e` signal for ALTECC_DECODER IP core. • Removed `sconst` port information as the port is no longer available for LPM_COUNTER IP core. • Provided an example to use the LPM_HINT parameter.
August, 2014	2014.08.18	• Updated parameterization steps for legacy and latest parameter editors. • Added note for IP cores that do not support Arria 10 designs. • Added device migration information.
June 2014	5.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog. • Added standard information about upgrading IP cores. • Added standard installation and licensing information. • Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.
June 2013	4.0	• Added **ALTERA_MULT_ADD (Multiply-Adder)** on page 6-1 section. • Removed the following obsoleted megafunctions: LPM_ABS, ALTACCUMULATE, ALTMULT_ACCUM, ALTMULT_ADD,

Date	Version	Changes
February 2013	3.1	• Updated Table 52 on page 63 to include Stratix V information for accum_sload port. • Updated Table 54 on page 65 to include Stratix V information for PORT_SIGNA and PORT_SIGNB parameters.

February 2012	3.0	• Added Arria V and Cyclone V device support.
		• Updated the parameter description for the following section:
		  • ALTMULT_ACCUM (Multiply-Accumulate)
		  • ALTMULT_ADD (Multiply-Add)
		• Added the Double Accumulator section.

July 2010	2.0	• Updated architecture information for the following sections:
		  • ALTMULT_ACCUM (Multiply-Accumulate)
		  • ALTMULT_ADD (Multiply-Add)
		  • ALTMULT_COMPLEX (Complex Multiplier)
		• Added specification information for all megafunctions

November 2009	1.0	Initial release.